A METHODOLOGY FOR THE DESIGN OF COMPLEX COMPUTER SYSTEMS IN AGRICULTURE AND AQUACULTURE

By

PHILIP A. FOWLER

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1998

ACKNOWLEDGEMENTS

I would like to first thank Dr. Ray Bucklin for his patience in my endeavor to make this project happen. His many consultations and directions made this a much easier project than might have occurred otherwise. It has been a pleasure to work with him for these last five years.

My committee members, Sencer Yeralan, Direlle Baird, Fedro Zazueta, and Ed Lincoln, I am sure have found me somewhat elusive, but I certainly have to thank them for their questions and answers. I give my appreciation for Bob Tonkinson's clever solutions to some rather tricky technical problems and for all those trips to where ever we had to go. I have a special thanks to Flavio Fialho whose developing expertise in the field of microcontrollers I could not have done without. I would also like to thank the Agricultural and Biological Department at the University of Florida who had to put up with me.

My personal thanks to my parents who wanted this much worse than I. I would lastly like to thank my wife, Maite, who may someday understand all of this.

TABLE OF CONTENTS

page

ACKNOWLEDGEMENTS ii

LIST OF TABLES vi

LIST OF FIGURES vii

ABSTRACT x

CHAPTERS

1. INTRODUCTION 1

2. LITERATURE REVIEW 5

Aquacultural Trends 8
Commercial Production 13
Development of Microcontrolled Products 14
Water Quality and Environmental Concerns 15
Nutrition and Feed 15
Combining Aquaculture and Microcontrollers 19

3. METHODOLOGY FOR DESIGNING COMPLEX COMPUTER SYSTEMS IN AGRICULTURE 21

Seven-Step Approach 21

4. APPLICATION OF METODOLOGY TO AN AQUACULTURAL SYSTEM 26

5. MODEL DEVELOPMENT 42

Growth Model 43
Dissolved Oxygen Model 44
Photoperiod Model 49
Waste Management Model 51
Flow Control 55

6. COMPUTER CONTROL SYSTEMS 62

Networking Microcontrollers 67
CAN 2.0A Protocol Basics 67
FP-CAN Protocol Basics 68
FP-CAN Protocol Messages 69
FP-CAN MOM CAN Figuration 71
General 74
CLINK Daughterboard 75
AGUA Daughterboard 76

7. MICROCONTROLLER HARDWARE DESIGN 77

8. A MICROCONTROLLED SYSTEM IN AQUACULTURE 87

Software Development 98
Interfacing the System 100

9. RESULTS AND DISSCUSSION 107

Growth Prediction 107
Control of Flow of Water 112
Control of Oxygen 115
Control of Temperature 118
Photoperiod Control 120
Filter Design 121
Reliability 121

10. CONCLUSION 123

APPENDICES

A SCHEMATICS 126

B SOFTWARE FLOWCHARTS 131

C MICROCONTROLLER CODE 146

D INTERFACE CODE 183

main.h 183
system_status.h 184
growth.h 186
_feed.h 189
_oxygen.h 194
_system_design.h 195

_temperature.h 196
err_F.h 196
filter_design.h 197
Graph_Output.h 198
photo.h 199
pump_error.h 200
species.h 201
waste.h 201
comport.h 203
_carp.h 205
Catfish.h 206
_Salmon.h 208
_Stripped_Bass.h 210
_Sturgeon.h 211
_Tilapia.h 213
_Tropical.h 215
_Trout.h 216
Main.cpp 218
system_status.cpp 224
growth.cpp 238
feed.cpp 248
oxygen.cpp 269
system_design.cpp 271
err_F.cpp 274
Filter_design.cpp 274
graph_output.cpp 276
photo.cpp 281
pump_error.cpp 286
species.cpp 286
waste.cpp 288
Comport.cpp 292
_carp.cpp 299
_catfish.cpp 303
_salmon.cpp 307
_Stripped_Bass.cpp 311
_Sturgeon.cpp 314
_Tilapia.cpp 318
_Tropical.cpp 323
_Trout.cpp 326

APPENDIX E. PINOUT AND CONNECTIONS 331

REFERENCES 337

BIOGRAPHICAL SKETCH 340

LIST OF TABLES

Table page

1. WorldWide Microcontroller Shipments (in millions of dollars)....6

2. WorldWide Microcontroller Shipments (in millions of chips)....6

3. Parameters affecting aquaculture....17

4. Distributive interaction of system....36

5. CAN 2.0A protocol....68

6. ID field breakdown....69

7. Bit settings for baud rate based on crystal frequency....73

8. System control of functions....106

LIST OF FIGURES

Figure page

1. Physico-chemical interactions in an aquatic ecosystem....16
2. Communications links in system....32
3. Basic biological model of a fish....42
4. Dissolved oxygen as a function of temperature....47
5. Waste model flowchart....54
6. PWM output with a low duty cycle....57
7. PWM output with a high duty cycle....58
8. Flow model flowchart....61
9. System response times....64
10. Block diagram of 80C592....82
11. Basic tank design....87
12. Settling filter....90
13. General overview of system....93
14. Opening screen....101
15. System status screen....102
16. Model vs Data....108
17. Model vs tank 1 data....109
18. Growth screen with early growth prediction....110

19. Growth screen with later growth prediction....111
20. Flow control section of system status screen....112
21. % Duty Cycle versus water flow rate....113
22. Fixed duty cycle versus flow rate....114
23. Real time graphical output....115
24. Dissolved oxygen calculator....116
25. DO readings versus the percent duty cycle....117
26. System temperature....118
27. Noise spikes in communication....119
28. Input screen for photoperiod....120
29. Photoperiod data....121
30. Differential pressure readings....122
31. Motherboard schematic....126
32. AGUA Daughterboard schematic....127
33. CLINK Daughterboard schematic....128
34. Header Schematic....129
35. Tee Schematic....130
36. Motherboard initialization....131
37. AGUA-feed flowchart....132
38. AGUA-con flowchart....133
39. AGUA-con flowchart continued....134
40. AGUA-con flowchart continued....135
41. AGUA-con flowchart continued....136

42. CLINK flowchart....137
43. CLINK flowchart continued....138
44. CLINK serial messages flowchart....139
45. CLINK serial messages flowchart continued....140
46. CLINK serial messages flowchart continued....141
47. CLINK serial messages flowchart continued....142
48. CLINK serial messages flowchart continued....143
49. CLINK CAN messages....144
50. CLINK CAN messages continued....145

Abstract of Dissertation Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

A METHODOLOGY FOR THE DESIGN OF COMPLEX COMPUTER SYSTEMS IN AGRICULTURE AND AQUACULTURE

By

Philip A. Fowler

December 1998

Chairman: Dr. Ray A. Bucklin
Major Department: Agricultural and Biological Engineering

A methodology was created for the integration of agricultural systems and computer control systems. A microcontrolled system was developed to demonstrate that the methodology could be used to design complex computer systems. The system was integrated through a distributed network and linked to a PC for human interface interaction. The methodology was developed as a systematic way to design a system that involves non-stationary dynamic processes. A seven-step approach breaks the system up into sub-models, analyzes each model, compares their interactions and then recombines them into a complete system.

The system was designed to complete an aquacultural grow-out cycle of the species Tilapia. An aquaculture model was chosen because of its varied parameters and harsh environment. The growing of fish presents a varied environment that needs constant

monitoring and control to maintain quality. The system consisted of three microcontroller units and a PC. A process of system development was used to combine both the control of the systems components and their interactions into one complete system. Software was developed to interface all of the units and the PC. The system was run under varying conditions and tested for failure modes. It proved to be adequate for most failures and adapted to conditions, as dynamics required. The seven-step methodology provided the systematic approach to the development of the complete process.

CHAPTER 1
INTRODUCTION

Technological advancements in computer control systems are being incorporated into agricultural processes. These systems can be simple processes on the component level but when they are combined, become complex and difficult to manage. Agricultural processes are dynamically changing systems. Computer control systems are an ideal choice to build automated methods involved the dynamic nature of the agricultural process. To create reliable computer control systems in agriculture, it is necessary to develop a methodology designed for that goal.

The world's fish stocks have recently peaked in production and are presently in a state of decline while the demand for such products has steadily increased. Wild fish regenerate at rates that are determined by nature. The ability of man to harvest has surpassed the ability of nature to replenish in many parts of the world including the Atlantic, Mediterranean and Pacific. This decline in the species of fish has led to a growth in aquaculture production. The total world catch is expected to top out at 100 million tons with approximately 70 to 80 million tons of edible fishing products. The expected demand by the year 2000 will be 100 to 140 million tons, thus a short fall of 20 to 70 million tons of fish that will have to be produced by other means. Aquaculture is the only means to make up this short fall (Pillay, 1990). World value of aquaculture production was 26.5 billion dollars in 1990, but the developed nations in North America were responsible for only 3.2 percent of the production (FAO, 1992). The

need to increase production in the United States is in conflict with growing concerns over water quality and adverse environmental impact. The development of intensive closed recirculating systems will help improve quality and quantity of locally produced product and lessens the environmental impact. At present there are very few profitable systems operating. This is due to three factors: (1) variable climate, (2) high labor costs, and (3) increased competition for dwindling water and land resources coupled with an unsympathetic regulatory bureaucracy. Intelligent control systems that are integrated into biological system can help improve the economics of aquacultural systems by improving the management, operation, and production of the system.

Food fish have not been successfully raised in closed aquaculture systems on a commercial level. This has been due to the cost of equipment, high management demands, large water make-up, high-energy demands, and the low prices given to the species cultivated. The advantages of the closed recirculating systems are higher density cultures, ability to locate close to markets, better control of disease and health, and timing of culture to market desires. The design of systems to meet the challenge of producing economical systems is of prime importance to offset the shrinking stocks of fish and the rising demand for the product.

Technical advances in electronics can make it possible to overcome some of the problems associated with aquaculture system design. Microcontrollers in specified areas of the systems can be made to do control of system parameters. The use of microcontrollers facilitates localized control of the system and reduces chance of catastrophic failure. When the microcontrolled units are networked together, the chance of over all failure of the system is reduced.

Control systems can be used to automate and monitor complex biological systems. A control system is a collection of devices that are interfaced to a biological system using sensors and logical programmed rules. The devices then use the information supplied from the sensors to regulate the system to optimum performance by controlling such functions as flow rate, feed rate, photo-period, oxygen, total ammonia nitrogen (TAN), CO_2, temperature, dissolved oxygen (DO), make-up water and so forth. Intelligent control systems are those that not only report and control certain functions but can make decisions to take corrective actions at a local level of control. A series of intelligent controllers can be collectively monitored by a central processor for overall system control but each local controller will be able to take independent action if it is severed from the system.

Networking microcontrollers is a cost-effective means of achieving more synergistic systems. When the control is distributed, greater flexibility occurs because the independent systems manage themselves until a system interaction is required. A robust network is required to always maintain over all system control. A central unit can be used to monitor the system and check for failures of any of the sub-systems.

Sub-systems consist of actual microcontrolled modules that are designed to do certain tasks and to notify the system of any out of range conditions. Modules can be asked for specific data or given commands to carry out a specific task. These modules do things such as monitor environmental conditions, test for failure, maintain pump speeds, change environmental conditions, deliver feed, adjust photo-period, and switch on/off devices, up to the limit of the I/O of the controller.

The objective of this dissertation was to create a methodology that simplifies the procedure of integrating computer control systems and agriculture. The dynamic nature of

biological problems in agriculture causes complex interactions that need a systematic approach when trying to be combined with computer control. A methodology is needed that is general enough to be able to be adapted to the many different types of problems that occur in agriculture. An aquacultural system was designed to demonstrate the development and application of the methodology.

CHAPTER 2
LITERATURE REVIEW

Computer control of agricultural processes is now becoming more common place. The use of computer controlled devices in industry is paramount in the growth of automated systems. An essential factor in the profitability of these systems is in automation. Agriculture in the US has become the world leader in productivity through intensification, mechanization, and automation (Lee, 1994). Embedded controllers will help automate aquaculture through better management and control. The current trend is toward the use of industrial process control systems composed of sensors/transducers, meters/transmitters, communication multiplexers, actuators/output devices, computer hardware and computer control software (Lee, 1994). The use of all these devices creates large interactive systems and when used in the control of non-stationary dynamic processes, the systems become complex. To reduce the complexity of such systems an embedded control system that is used on a distributed network can be utilized. In order to create these types of systems there is a need for a methodology to be used so that the design of such systems is precise.

Embedded controllers were first developed in the early 1980s and were used for very specific tasks in industrial control systems. The requirement to manage information processing in today's systems applications necessitates the implementation of embedded control. The use of embedded control allows the system designer to develop a solution for the specific

application with the least amount of overhead cost and space (Microchip 1993). The growth of the microcontroller industry has been and will continue to be at a fast pace with many new applications being found. The following tables show recent trends in microcontroller growth and their expected future growth.

Table 1. WorldWide Microcontroller Shipments (in millions of dollars).

	'90	'91	'92	'93	'94	'95	'96	'97	'98	'99	'00
4-bit	1393	1597	1596	1698	1761	1826	1849	1881	1856	1816	1757
8-bit	2077	2615	2862	3703	4689	5634	6553	7529	8423	9219	9715
16-bit	192	303	340	484	810	1170	1628	2191	2969	3678	4405

Source: WSTS & ICE - 1994(FAQ, 1995)

Table 2. WorldWide Microcontroller Shipments (in millions of chips).

	'90	'91	'92	'93	'94	'95	'96	'97	'98	'99	'00
4-bit	778	906	979	1036	1063	1110	1100	1096	1064	1025	970
8-bit	588	753	843	1073	1449	1803	2123	2374	2556	2681	2700
16-bit	22	38	45	59	106	157	227	313	419	501	585

Source: WSTS & ICE - 1994(FAQ, 1995)

In order to use an embedded controller, software must be written that will fit the controller to its application. A major goal of intelligent control systems is to achieve high performance with increased reliability, availability, and automation of maintenance procedures (Polyoarpov and Vemuri, 1995). Since conventional control schemes are limited in their range of practical applications, fuzzy logic control is receiving increased attention for intelligent control applications (Kim et al., 1995). Fuzzy controllers can be made to learn their control process and adjust accordingly to their environment. A common method for this "learning" is

by the use of adjustable look-up tables. An initial algorithm is created in which the maximum parameters are listed in the look-up table with results given that control the system in some known state. When a certain error is found an adjustment to the table is made and the result of the correction that is made is detected. If the result is more favorable to the environment, the table is left in that form. Should the result create a greater error, another scheme is tried and results are checked. This is repeated until the best solution is "learned" and the table is adjusted. The algorithm must look for oscillations that may arise and be able to damp them out. The controller can then continue to monitor its system and adjust as the conditions change.

A microcontroller is an entire computer manufactured on a single chip (Yeralan and Ahluwalia, 1995). This computer includes all the necessary peripherals such as CPU, memory, I/O, timers, interrupt controller, and communications to be able to be interfaced with hardware to act as an embedded control system. Due to their small size, microcontrollers can operate at low voltage and current consumption and are very energy efficient. Microcontrollers are used in many applications such as automobiles (ABS braking systems, engine control), appliances (microwave oven, refrigerators, televisions, VCRs), auto-focusing cameras, environmental control (greenhouse, factory, home), intelligent sensors, and many other uses. Microcontrollers are very adept at doing local management of specific systems and have the ability to communicate with other systems to share data and resources.

There has been a paradigm shift in the computing industry from monolithic computers to client-server systems. This shift moves the computing power to where it is needed. This type of distributive computing is the current trend and is seen in the use of the Internet and intranets (Greenberg and Lakeland, 1997). There have been significant technological improvements without the ability to capitalize by their implementation. Technology has evolved to where we

understand the components but when we put the components together into a system, it becomes complex and difficult to comprehend.

Methodologies designed for the development of computer control strategies are not very common and do not exist for adapting agricultural based applications. A methodology is by definition a set or system of methods, principles, and rules for regulating a given discipline, as in the arts or sciences (Random House, 1996). When applied to the combination of two disciplines it requires a general description of the solution to a broad-based problem. The goal of such a methodology is to provide a reliable system that will perform its intended function for a specified interval under stated conditions. Forrester was one of the first to elaborate on such systems development (Forrester, 1968).

Aquacultural Trends

Consumption of aquatic species in the United States (US) has been on the rise as the diet conscious population has been consuming greater quantities of product, especially fish. World value of aquaculture production was US$ 26.5 billion in 1990 but North America was responsible for only 3.2 percent of the production, down from 3.9 percent in 1989 (FAO, 1992). The US is the worlds leader in agricultural production but is a major importer of seafood. At least 50 percent of the seafood that Americans consume is imported. Within 15 years, fish farming and sea ranching could provide nearly 40 percent of all fish for the human diet and more than half of the value of the global fish catch, says a new report by the Consultative Group on International Agricultural Research (CGIAR, 1995).

The US, once the leader in world fisheries, now ranks sixth in total volume of products landed, behind Russia, Mainland China, Japan, Peru, and Chile. The world is currently harvesting at or near the maximum available tonnage from the earth's supply of wild fish

stocks. Freshwater fish culture, or aquaculture, cannot meet the domestic demand for fish products and the United States has become the worlds leading importer of fishery products, thus adding to the deepening national deficit (Ericsson, 1995). This has led to a trade imbalance that is uncommon in the agricultural sector of our economy.

The main reasons for the shortage of production facilities in the US is the lack of suitable water supplies and the variable climate. As land based sites become more toxic from the effects of pollution from pesticides, herbicides, insecticides, fertilizers and other unfriendly substances, there is a greater need to develop closed systems that are free of these influences. The availability of clean plentiful water, for use in aquaculture, is rapidly diminishing so it is necessary to further develop low water use aquaculture systems. The imposing of strict environmental laws governing water usage and effluent disposal has created an opportunity for closed recirculating systems in aquaculture. The cost-performance relationship of the current systems has made these systems unprofitable except for high value species.

Aquaculture is the managed growth of aquatic species under controlled conditions. Aquacultural engineering is the use of engineering principles in the culture of aquatic organisms (Wheaton, 1993). The increase in demand has made the economics of engineered closed recirculating systems begin to come close to the profitability side of the market place. The use of embedded controllers and intelligent control is a recent development in the field of aquaculture. The integration of microcontrollers and aquaculture should improve the ability to manage and control aquaculture systems. This type of embedded control system would work well in the rapidly changing aquacultural environment.

In aquaculture, there is a need to manage the systems on a continuous basis. This management is highly labor intensive and must be concise, reliable, and timely. The delay of a

few minutes can lead to the loss of a valuable crop. The use of embedded controllers can increase the productivity of a system by doing the monotonous jobs of continuous monitoring and control. The simple task of keeping track of water level by use of an intelligent sensor can give the operator valuable information as to the trend of water level which might indicate a slow leak in the system, and thus preclude a disastrous extreme low level. The biggest advantage of embedded controllers is their cost advantage over redundant computer controlled systems. Since an embedded controller can function as a stand-alone device it can independently sound alarms or controls systems without the need to be centralized to a computer. The failure of any independent system will not lead to the failure of the overall system. This allows the system to continue to function, even if not at full efficiency, and maintain the biological system. The cost advantage arises from the fact that many small-embedded controllers can be purchased for the same price as a single computer. Many problems in aquaculture are site specific so that technology must adapt to those localized problems. Microcontrollers offer the ability to adapt to the specific problems through their ease of programmability and the ability to adapt control functions to the site-specific problem. Large-scale solutions are often expensive and inflexible where as small-scale solutions offer the ability to solve small problems with small costs.

Intelligent control systems are presently being used to control the feeding schemes of aquatic organisms. The precise control of feed input to an aquaculture system is of prime importance. Feed not consumed within hours decomposes, decreasing available oxygen and other water quality factors (Hatch and Kinnucan, 1993). The controller interfaces with electric motor driven feeders to allow for precise feeding amounts. The controller can also detect problems with the local system and report back to a central processor to notify the operator of

problems. The local intelligence of the controller also will stop the feeding process should other emergencies arise such as power failure or out of parameter conditions such as temperature being out of range. The integration of the feeding scheme into the systems quality standards will maintain a more productive system.

An embedded system was designed and tested by the author, as part of a previous project, that did automated backwashing of a sand filter (Bucklin, 1995). The system consisted of a differential pressure sensor that was used to detect loading on the sand filter. When the pressure exceeded a certain limit a series of valves were either opened or closed in order to effect a backwash of the filter to remove organic buildup. The system was tested both with this pressure detection and with regular timing intervals of backwashing, showing the flexibility of the microcontroller. The system is a scheme of operation that could be changed by simply making a few changes in the software and down loading to the controllers memory. Although the controller system worked flawlessly, there was a problem with the valve actuators that caused problems in flow control of the waste effluent. With better valve actuators, this system could be very useful in aquaculture.

The use of embedded controllers has been tested in other agricultural systems. A microcontrolled system designed by the author and installed in a dairy-milking barn (Bray et al., 1998). The purpose of this system was to monitor vacuum levels in the pulsators of milking machines. The system used a network of microcontrollers to measure each vacuum line attached to the cow and to send this data back to a PC where it was logged.

It is hard to quantify the impact of embedded controls although they can greatly increase the quality of the system and decrease the man-hours needed to do some of the monotonous tasks such as feeding. The improvement in the quality of the feeding scheme can

be seen by the repeatability of amount and in the ability to increase frequency of feeding that should lead to a higher growth rate. The fact that a farmer can rely on the embedded controller system to continue to give service to the system and not have to rely on questionable farm labor will outweigh the higher initial cost and result in the system eventually paying for itself. Since crop loss can occur in very short periods of failure with important system components (air pumps, water pumps, loss of water) a reliable system must be implemented to notify the operator. The independence of embedded controllers gives this extra reliability.

The development of an integrated system to control a process involves the use of models to create the logic that is used for control. A model is the building of an analog of a system being studied. Model development in agriculture has been going on for some time but the development of models specifically for aquaculture is new. Aquaculture is characterized as a relatively underdeveloped and yet poorly defined technology (Allen, et al 1984). The use of models in an aquaculture process is restricted to simplistic models so the process can be carried out. This is not a detriment to the process control because in aquaculture, processes are somewhat slow and exact control is flexible. Many models can be borrowed from other fields and adapted for use in this process such as solar models or oxygen exchange models. There are models that must be developed to fit new processes as they are created, such as a model to remove waste in a new type of filter. The growth model for aquaculture takes many forms due to the dynamic nature of the growth of the species being used. There does not appear to be one model that will fit the complete life cycle of most fish. The biological sub-model is usually the most difficult to formulate because of the complexity of a biological organism and its interactions with its environment (Allen, et. al, 1984).

Commercial Production

The raising of fish in closed aquaculture systems has not been successfully done on a commercial level. There are relatively few reports of profitable recirculating production systems in operation (Timmons and Losordo, 1997). This has been due to the cost of equipment, high management demands, large water make-up, high-energy demands, and the low prices given to the species cultivated. The advantages of the closed recalculating systems are higher density culture, ability to locate close to market, better control of disease and health, and timing of culture to market desires. The design of systems to meet the challenge of economical systems is of prime importance to offset the shrinking stocks of fish and the rising demand for the product.

Technical advances in electronics can make it possible to overcome some of the problems associated with aquaculture system design. Ebeling, Whitson, and others have done much of the work of laying down the schemes needed for computer control of aquaculture systems (Ebeling, 1994;Whitson et al., 1997). Microcontrollers in specified areas of the systems can be made to do control of the specified system parameters in place of the more expensive computers. The use of microcontrollers facilitates localized control of the system and reduces chance of catastrophic failure. When the microcontrolled units are networked together, the chance of over all failure of the system is reduced.

Networking microcontrollers is a cost-effective means of achieving more synergistic systems. When the control is distributed, greater flexibility occurs because the independent systems manage themselves until a system interaction is required. A robust network is required to always maintain over all system control. A central unit can be used to monitor the system and check for failures of any of the sub-systems.

Sub-systems consist of actual microcontrolled modules that are designed to do certain tasks and notify the system of any out of range conditions. The module can be asked for specific data or given commands to carry out specific task. These modules do things such as switch on/off devices, monitor environmental conditions, test for failure, maintain pump speeds, change environmental conditions, deliver feed, adjust photo-period, and other tasks up to the limit of the I/O of the controller.

Development of Microcontrolled Products

The design of a microcontrolled product begins with an understanding of what problem is being resolved. The problem is then matched with a particular type of microcontroller that has the necessary peripherals to do the task. There are many manufacturers of microcontrollers each with a slightly different composite of features. There has been some standardization in the field with Intels development of the 8031 microcontroller. Many other manufacturers make this chip with variants in design. The common attribute of the chip is the pin layout and the code used to program the chip. Each type of chip has some specific functions that are better or less suited for the job. Some of the typical variations are A/D converter, communications ports, multiple inputs and outputs. "The move from an industry based largely on extensive culture to one utilizing computerized, intensive systems has been inhibited by the slow development of aquaculturally oriented software" (Rusch and Malone, 1993, p.53).

The monitoring and control of an aquaculture system is certainly less complex than many industrial systems in use today. The equipment used for interaction with aquatic systems is readily available but the combination of equipment and software has not been thoroughly implemented. This software must be flexible enough to accommodate the many

different species that are cultured and must be able to interface with a variety of different system combinations.

Water Quality and Environmental Concerns

An aquaculture system contains diverse groups of biological organisms that interact to create a number of complex physiochemical reactions. In Figure 1 (Lawson, 1995) these interactions are shown. Each of these reactions is difficult to describe when combined with the other reactions taking place. In the development of a closed recirculating system, there is a need to establish the parameters that will be followed in the system. Fish vary in their environmental requirements depending on the type of species that is cultivated. The temperature of the water will be an important criterion as to whether the species selected can be cultured on site (Pillay, 1990). The temperature that the species must be cultivated at is usually a narrow range for optimum growth. Some of the other primary concerns for cultivation that will influence the design of an aquaculture system are shown in Table 3 (Lawson, 1995). Each of these parameters must be accounted for in a systems design. The range of these parameters will have an effect on how well the culture survives and grows. Some parameters are more critical than others and must be kept within a narrow range for the optimum growth to occur.

Nutrition and Feed

In order to produce high quality fast growing fish, it is necessary to develop a plan that includes the proper nutrition for that purpose. In modern aquaculture, feeding is one of the major elements of cost of production and may amount to 50 percent or more (Pillay, 1990).

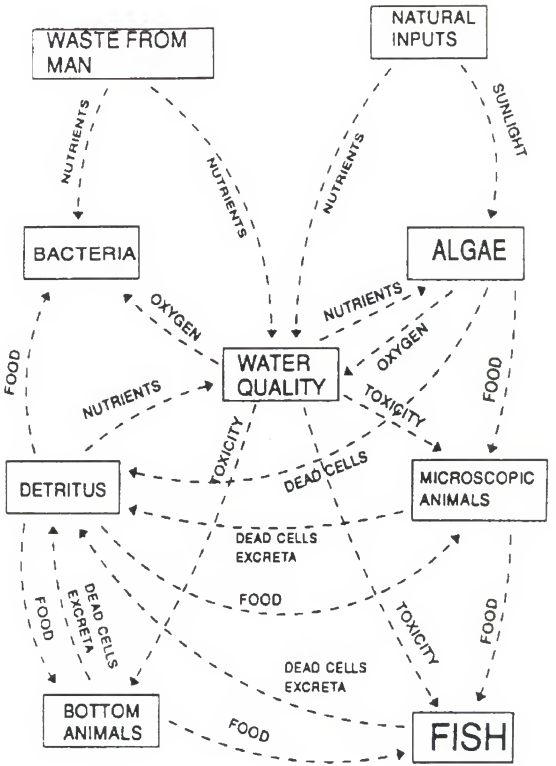


Figure 1. Physico-chemical interactions in an aquatic ecosystem.

Table 3. Parameters affecting aquaculture.

Parameter	Concentration
Dissolved Oxygen	5 to saturation
Ammonia (NH_3)	<0.02
Ammonia (TAN)	<1.0
pH	6.5-8
Nitrate (NO_3)	0-3.0
Nitrite (NO_2)	<0.1
Alkalinity	10-400
Salinity	<5%
CO_2	0-10

Thus, selecting the proper feed and feeding scheme can influence the economic viability of a system. In closed recirculating systems it is necessary to provide all the nutrients that the species will require and at the proper times. This can best be done by automating the system so that there are reliable methods of feeding both the correct amount and at a regular schedule.

Digestion is the conversion of proteins, carbohydrates, and lipids from macromolecules to sizes that can be utilized by the body and absorbed into the bloodstream. Proteins are converted into amino acids or polypeptide chains, carbohydrates into simple sugars, and lipids into glycerol and fatty acids. There is a certain conversion

rate for each of these materials and the indigestible materials are eventually voided as feces.

High quality fish feeds contain carbohydrates in the amounts of 20 percent for cold-water fish and 30 percent for warm-water fish. Protein is the main source of nitrogen and essential amino acids. For maximum growth, young fish require from 40 to 60 percent of their diet as protein.

Temperature affects the protein uptake and assimilation. The most favorable temperature for a given species is the one at which the difference between maintenance requirement and voluntary food intake is greatest and at which optimum efficiency of growth occurs. Fish are some of the most efficient animals at converting feed energy into protein.

Lipids are a group of fat-soluble compounds from the tissues of plants and animals and are mostly fats. They contain more energy per unit weight than any other biological product, approximately 8.5 kcal metabolizable energy per gram. The lipid content of commercial aquaculture diets is less than 10 percent due to problems associated with processing.

In commercial aquaculture, the main supply of feed is from processed feeds in order to achieve a balanced diet and create maximum growth. The type and composition of the feed used in an aquaculture system will greatly influence the ability to produce maximum growth. This formulation changes with the different growth stages of the culture. The environment can also influence the type of feed used. The type of formulation used will also influence the environment since that which is not metabolized is excreted into the living environment and must be dealt with by removal or neutralization. The feed that is

metabolized is then excreted as byproducts that increase the waste level. The other important aspect of feeding is the method and timing of the feed distribution. Ideally, feeding should be based on the dietary requirements or the metabolized energy value of the feed. Typically, the level of feeding is determined as a percentage of body weight. Microcontrollers and their computational abilities make a precise approach to feeding and the timing of such feeding a possibility. Studies on the effects of feeding frequency on the growth of fish indicate that there is an optimum frequency, above which additional feedings produce no advantage (Pillay, 1990). This feeding ratio must be adjusted for temperatures affects on the metabolic rate of the culture. With the advent of new technology, many aquaculture facilities are using computer-controlled automated feeders that are regulated by programs that take into account the dynamic environment of the culture.

Combining Aquaculture and Microcontrollers

In literature, a clear task has been presented to research for the development of new technology to increase the productivity of commercial aquaculture production. The demand for high quality aquacultured products and an increasing concern for resource conservation has led individuals and large corporations to invest time and money in commercial scale recirculating production systems (Timmons and Losordo, 1994). With the advances in technology, recent developments make it possible to achieve this goal. Microcontrollers have been in production since 1980. They have been put to many industrial applications since that time. These devices make many labor intensive and redundant tasks a simple matter to deal with. The introduction of these devices into agricultural applications is a task that requires a precise methodology to achieve this goal.

The process that is created in a closed recirculating aquacultural becomes complex and unmanageable unless a systematic approach to development is used.

CHAPTER 3
METHODOLOGY FOR DESIGNING COMPLEX COMPUTER SYSTEMS IN AGRICULTURE

The design of control systems in agriculture is usually based on experience and fundamental engineering principles. Emerging computer technology emphasizes the need to formalize a methodology for developing such control systems. This study proposes an approach consisting of a seven-step method, based on the formulation of a system into its sub systems, which are then analyzed and designed into a complete system. Software is developed that will make the integration of these subsystems compatible with each other. Each subsystem should be able to act independently in its scope and communicate effectively with the rest of the system

The Seven-Step Approach

1. Describe the overall system objective. This is a description of what the process will be producing or controlling. A detailed description is not required at this point, only the objective of this process. It should be made clear what the inputs and outputs of the system will involve.
2. Break the system into independent functional subsystems. The subsystems should be described in detail with all inputs and outputs. Each of these systems will be a complete process that can operate independently although the system may depend on other systems for its inputs and other systems may depend on the output from this system.

3. Define the processes of each subsystem, describe how each subsystem works, and identify the parameters and variables associated with each subsystem and each subsystem process.

   Arrange the parameters into the following categories:

   - Observed - parameters that can be physically measured e.g. dissolved oxygen level.
   - Control - parameters, which can be altered by a controller, e.g. pump speed.
   - Hidden - parameters, which are not easily measurable or apparent but influence the system, e.g. biological stress upon organisms.
   - Independent - parameters which are beyond the control of the system, e.g. time.
   - Dependent - parameters that may be described by functions of other system parameters, e.g. heat flow per unit time.

4. Develop or use an existing model for each subsystem process. Decide if it will be a procedural model or an inferential model.
5. Define what interactions take place between the subsystems and describe the subsystem processes and subsystem parameters that are involved in such interactions. It is important to identify those subsystems that are dependent on other subsystems. The dependent subsystems should be listed in order of importance with the least dependent at the top and the most dependent on the bottom. This list gives a failure mode in which the system can be analyzed for importance of subsystem function.
6. Involved in subsystem interactions, categorize the exchange of the following:

- Materials- these are the physical components of the system that are moved through the system and upon which each process interacts to produce some output or input to the next subsystem process.
- Energy- the energy required at each level must be balanced or accounted for during its process and should sum to zero unless energy is the product. Imbalances in forces between subsystems can lead to a disruption of flow in the process.
- Information- the exchange of information between subsystems will assure that all processes work together in a homogeneous manner. This information will give the necessary values for the setting of levels or parameters that each subsystems must have to function. The requirements of each subsystem to have the correct information are determined by the importance of that subsystems process. It is important that all the information from the subsystems be centralized so that some complete system status can be determined.

7. Integrate all of the subsystems into one system. This is the design stage in which the system is put together with the following considerations:
    - requirements of the system
        - Describe or quantify system behavior. The system behavior is often more complex than the cumulative behavior of its subsystems. The nature and level of the interactions among the subsystems cause this synergistic effect. The system needs to be checked for interactions that can cause problems to arise. The synergistic interactions can be both good and bad.

- Describe or quantify system reliability. System reliability must be ensured by the use of redundant systems. Repeatability must be shown with the system functioning. Self-diagnostic algorithms should be incorporated in the design stage to facilitate rapid prototyping. Technologies should be picked that are suitable for this system.
- Describe system error detection and correction requirements and capabilities. The systems should be smart with the ability to determine fault conditions and to take corrective actions. These systems should learn from past history by data logging the important parameters and then analyzing the data to set up better control schemes. The subsystems should show independence from each other so that they can act alone to solve their immediate control problems. They should communicate these parameter considerations to a central processing center to coordinate the process control.

- requirements for communication between subsystems
- requirements of levels of computing measured in million instructions per second (MIPS). This is the amount of instructions that can be carried out in one second. In the typical information processing equipment, e.g. mainframe computer, personal computer, micro controller, simple logic control, switches, manual control, the speed required may be in other units
- software requirements
    - high level language (BASIC, C, Integrated packages)
    - Low level language (ASM , ladder logic, Cross Compilers)

- Interface routines (Human/Machine, Device/Sensor)
- Communications protocol

The system should then be ready to be assembled and tested for reliability and function.

CHAPTER 4
APPLICATION OF METHODOLOGY TO AN AQUACULTURAL SYSTEM

The Seven-step approach is now given in more detail. This approach to the formulation of this process is now described in a step by step procedure.

1. The description of the control system should be set forth as an objective goal for the system. This is the process the system will control and what the outcome of this process is. The overall view of the system should be described in general terms so that there is a clear objective of what the process hopes to accomplish. Examples of this would be an aquaculture system for the grow-out of tilapia from fingerlings to marketable size, development of a monitoring system for pulsators in a diary barn, or creating a greenhouse system for growing flowers.

   The system that is described in the following work is an aquaculture grow-out system for the production of tilapia fish culture. This system will be fully automated with the use of microcontrolled subsystems that are integrated together through a distributed network. A PC gathers data and runs an expert system that acts to modify the controller parameters as the system works and monitors the system. The final output of the system is tilapia in the range of one kilogram.

   The system process involves maintaining water quality to an optimum degree for good growth of fish, providing nutrients for the culture, regulating the photoperiod, and removing waste material. The system must communicate between subsystems and

to the main PC by way of a distributed network. The system must be robust enough to handle failures of equipment and notify the operator when maintenance is required.

2. The reduction of the process into subsystems must be done so that each subsystem is an autonomous system capable of being modeled or described in detail. This subsystem should have input and output as its functionality goals. The process itself should be as simple as possible and still be able to maintain its objective. The system may depend on other subsystems for input but should be able to control its own processing towards its objective. The process description of the subsystem should be complete in its objectives, inputs, outputs, and control algorithms. In this example the subsystems are as follows:

- information
- environment
    - water quality
        - temperature
        - oxygenation
        - filtration
            - solids
            - biological
        - flow through the system
    - photo period
- physical containment
- nutrition
- communication

3. The subsystems and their processes should be concise and descriptive enough so that they can be thoroughly explained in a manner that allows each component or parameter to be analyzed on the lowest level. Once a parameter is defined, it can then be related to the model. The subsystems of the example are now described.

- Information- the need for information is apparent in any process but the acquisition of this information is the key to this example. The ability to match the species requirements to the system processes has to be done based on input of the required parameters that match the species. A database of species information is one subsystem in the system. This allows the operator to start the system with the necessary values for the different parameters based on species selection. Another information process is the logging of data in the system. This information is then used to recognize patterns that develop for the system and its various subsystems and is then used to compensate for local dynamic changes in the process.
- Environment- the stating of environmental conditions for this system is done by the breakdown into subsystems of each specific parameter.
- Water quality is further divided into minimal subsystems as follows:
    - Temperature- the control of an optimum temperature is required for maximum growth. There is a temperature range in which the fish will grow which must be maintained. There is an optimal temperature that the fish will grow the fastest and there are temperatures that may be required for spawning or other development. Actions must be implemented should the temperatures fall outside the required ranges. The subsystem of temperature control must be defined to match the required growth patterns desired. This system involves measuring the temperature

in the water and the addition or removal of heat to the system to maintain that level.

- Oxygenation- the subsystem of maintaining the oxygen level in the system is one of the most important in the system. Fish will quickly die when this level falls below a specified level for any length of time. This importance means that redundant systems should be used to maintain this level. This redundancy should be in both the monitoring and in the delivery. Oxygen is maintained in a system by the pumping of air into the water and the reliance of the exchange properties of the two media. The system involves the measurement of O_2 in the water and the rate at which air is put into the system.
- Filtration- filtration can be broken down into two types, solids control and biological control.
    - Solids- the build-up of solids from the introduction of feed into the system is a major problem for aquaculture. The feed is converted into feces by the fish and feed is left over from non-consumption. These particles then degenerate the quality of the water by further breaking down into ammonia that is toxic to the fish. The removal of solids is important in the maintaining water quality. The removal of solids is a continuous process but has peak times at which it should be accelerated. This system involved the timing of solid build up and the removal by mechanical means.
    - Biological- biological filtration is required to remove ammonia and nitrite. The process involves the conversion of ammonia to nitrite and then the conversion of nitrite to nitrate. A biological filter consists of a media on which certain

types of bacteria grow These bacteria are responsible for the conversion process. There are many different types of bio-filters in use. The one chosen for this system was a fluidized bed filter. This system utilized a sand media that needed a minimum flow rate to be fluidized and a maximum rate, at over which the sand blew out of the filter. Flow control was critical to this system. The design and selection of this filter also required calculation of surface are for the media to match the output rate of ammonia from the tilapia.

- Flow through the system- the flow of water through the system impacted the ability of the bio-filter to function and the removal of waste from the system.

- Photo period- the need to regulate the light period for the fish is important for physiological development. In nature, a circadian rhythm exits for any part of the world. This program is able to use the latitude and longitude, the day of the year and the time of day and determine if the light should be on or off. The only control function of the system was to provide an on or off condition for the light.
- Physical containment- the fish are contained in tanks which give them their water environment. The containment system consists of the tanks, plumbing, and filters. The design of the environment needs to be incorporated into the system so that those parameters that are associated with the containment can be accounted for. The total volume of water, the distance that the water is moved, and the friction in the system must be known for other calculations in the system.
- Nutrition- In a closed recirculating system the control of the flow of nutrients into the system governs what the behavior of the system will be. The feed introduced will cause solids to build up in the system. These solids are created when the fish eat the

feed and grow. The feed is digested and excreted into the water and some feed will not be eaten and cause an increase in solids. If the feed can be introduced in a systematic manner, a reduction of amount solids build up will be beneficial. The model for feed introduction can be based on a growth curve that is appropriate for the culture. This system is tied to a growth curve and the determination of feeding characteristics of the control system. The control model of the feeders is a sequential feeding scheme with a time base for each feeder.

- Communication- the ability of the system to function depends on how good the communication is between all of the subsystems, the PC, and the operator. A communications model is based on timing between the systems and includes events that can occur unexpectedly. A priority must be established for each level of communication. This system has several levels of priorities. The primary level is operator intervention. The operator has the ability to intercede at any time and change the system parameters. There is a level of communication between the PC and the operator. The operator may need information from the PC to make decisions for parameter settings. The PC is responsible for communication with the operator and the computer link. The computer link is responsible for the communications between the PC and the controllers. The controllers are responsible for the communications between the inputs and outputs that are involved with control of the system locally. Figure 2 gives a graphical description of how the links interface with each other. There is one output for the operator, which is the monitor. There are two inputs for the operator, which are the mouse and keyboard. The computer link has two way communications with both the computer and the microcontrollers in the system. The

microcontrollers take inputs from the sensors and have outputs to control various functions of the system, such as pumps, lights, heater, and feeders.

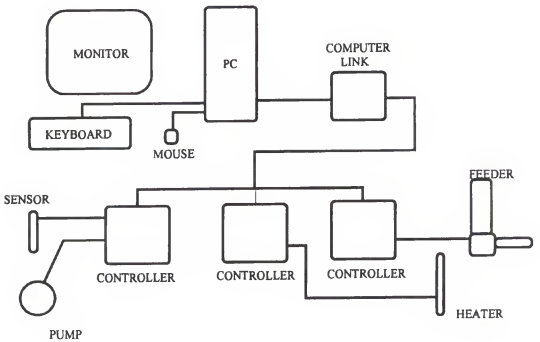


Figure 2. Communications links in system.

The variables and parameters are now defined and classified for each sub system.

- flow of water through the system
    - Water-water has various properties that can change with temperature and purity. The density and viscosity both rely on these parameters for their values. In this study, the flow through the bio-filter is the important limiting parameter. This leads to both a hydraulics and biological model as both aspects determine the proper flow rate.

- Elevation- the energy required to move water through a system is directly proportional to the change in elevation and the mass moved.
- Pump- the pump is the prime mover of the water through the system. It has energy requirements and efficiency ratings. The pump in this system is a controlled variable; that is, its power input can be varied which in turn varies the flow rate of the system.
- Plumbing- the pipes, tanks, and filters that are in the system direct the transfer of water through the system.

- Temperature
  - Water mass- this is the primary temperature concern in the system. The water has a mass that is definable by its volume and density. The amount of energy required to change the temperature of this mass is proportional to the temperature gradient and the mass.
  - Heater- this is the input device used to add heat to the system. It should be over designed so that any unforeseen occurrences can be accounted for.
  - Surface area- this is the total area of all surfaces that are in contact with the water or the air.
  - Insulation- this parameter is dependent on the type and thickness of material used.
- Dissolved oxygen
  - Water-water has the ability to hold dissolved oxygen at certain limits based on parameter temperature, density, salinity, and vapor pressure.

- Temperature- a controlled variable which can raise or lower the level of O_2 gas that the water can hold.
- Salinity- the salinity of the water may or may not be controlled and has a direct effect on the ability of water to hold O_2.
- Density- is a dependent variable based on temperature and solids content of the water.
- Vapor pressure- is an uncontrolled variable, which is dependent on the atmospheric conditions and altitude.
- Pump- the pump is a controlled variable, which can be changed to adjust the amount of air (O_2) being injected into the water.

- Feeding
  - Biomass- this variable changes with time and must be predicted by the system. It is the mass of fish in the system.
  - Feed quality- this is the type and make-up of the feed. It includes the protein content, vitamin additions, and various levels of nutrients that are made available to the fish.
  - Timing- this is a controlled variable, which is set by the operator and controlled by a sequence of operations.
- Photo period
  - Timing- this requires the calculating of sunrise and sunset for a particular place on the earth.
  - Light- is a controlled variable with only two states, on or off.
- Waste water management

- Timing- the timing is a controlled variable, which is based on desired exchange rate and capacity of the system.
- Pump- the pump is controlled by its flow rate.

4. The creation of an agricultural model is based on more practical and selective methods that lead to optimization techniques that tend to optimize the control variables based on real-time rather than the overall value of time. There are two types of models that can be made to utilize computer-processing methods. One is procedural and the other is computational. The modeling of the subsystem should be done in a way such that it will convert to a systematic process that can be formulated into sequential computer code. The speed of modern processors allows a more real-time approach to occur that lets parallel processes occur which appear to be simultaneous but are in effect sequential. There are some cases where parallel operations can occur. This happens with certain timers and Interrupt Service Routines (ISR). These routines can be triggered by independent hardware that is located on a microcontroller. The timers invoke their ISR by matching registers in either a count down or count up method. The routines may also be invoked by using the timers as counters, which can record the number of occurrences of an event. Another type of independent occurrence is an external interrupt, which is merely the changing of state on one of the ports, which will invoke an ISR. The timing of these events is in parallel and all operations but the ISR are sequential events. The model that is developed for the subsystem should have very specific goals in mind as the regulation and control of specific processes is what microcontrollers do best. The subsystem model should be reduced to the

least amount of both input and output variables that will fulfill the desired process. The model development of each of these subsystems is described in detail later in the work.

5. The interaction between the subsystems is mainly one of passing parameters so that control levels can be maintained. Each subsystem is designed to be as independent as it can be without outside input, although this is not the case for all systems. This system in this study consisted of a PC and three microcontrolled devices (AGUA-con, AGUA-feed, and CLINK). Table 4 shows how the various subsystems interact and the system goal of each subsystem. The ability of the system to communicate among the various controllers allows this interaction to take place. Whenever there is a need for outside input to the controllers subsystem this is an interaction event. The primary interaction, taking place between all subsystems, is the reporting of data from sensors.

Table 4. Distributive interaction of system.

	PC	AGUA-con	AGUA-feed	CLINK
DO		C	M	I
Temperature		M/C	M	I/C
Feeding	C		I/C	I
Flow Rate		M/C		I
Photo-period	C	I		I
Waste Removal	C		I	

M - measure IM - input manually C - controlled I - interaction

In the system, described here, interaction is made less important for some of the subsystems such as the control of flow through the system. The model that controls flow acts on its own and is unaffected by other sub-model parameters. This is not to say that the control scheme is independent of the total system as operator intervention can change the values that affect the rate of flow. The feeding scheme is completely dependent upon the PC to coordinate the feeding times. The controller does the actual

timing of the feeding and motor control but it is directed by input from the PC. The PC in turn is directed by the operator as to how the timing schedule is made and then incorporates a predictive model to decide on the daily rationing of the feed amount. The photoperiod model is a predictive model that is set up based on certain parameters such as location and day of year. The output of the model was a set time for sunrise and sunset. These times are then communicated to the controller responsible for turning on and off the lights. There is no operator intervention in this scheme other than deciding the initial parameters of location and time. Temperature control is a simple feedback loop of measuring the temperature and the deciding if the heater should be on or off. The scheme is made more complex by calculating the heat loss of the system, knowing the required temperature range and then calculating the time required for heating of the water based on the ability to heat the water. This model is not used but would have been useful in detecting impending failures of the system should the outside influencing temperature go beyond a certain range. Lack of funds was responsible for this oversight.

6. In systems that interact with each other there is a need to check what transfers of materials, energy, and information occur and how these interactions affect each other. The part of the system that involved the PC had only the transfer of information. This transfer of information is a critical function of the system. Should this exchange of information stop or be disrupted, the system would not function as designed. The advantage of a distributive system is that although a major part of the system might fail the system would still operate as discrete controllers and those controllers could then try to adjust the system in a failure mode. An example of this would be, should

communications from the PC fail, the controller which regulates the flow in the system will continue to maintain the pump at it last known set point. Information exchange between the different parts of the system was a continuous process that relied on many different types of communications protocols. The acquisition of data from the sensors is accomplished by three different methods, digital, analog, and serial communication. The digital method involved detection of the presence of a voltage signal. An example of this is the counting of pulses from the flow meter. The signal is counted over a time and then converted to a digital equivalent. The analog method detected a level of voltage present and then converted this signal to a digital equivalent. An example of this is the detection of the oxygen level in the water. The DO meter used a probe that generated a low voltage signal based on the concentration of oxygen in the water. This signal is conditioned to a 0 to 1 volt level that is directed to the analog to digital input of the controller. The serial communication method is utilized by the temperature sensors. These sensors had internal circuitry that converted the temperature to a digital signal and sent this signal to the controller by a serial interface.

7. The integration of all the subsystems into one large system must be done so that there are no conflicts that will keep the overall system from functioning. Integration involves connecting the outputs from each subsystem into a unified system.

The system description will define how it works. The aquaculture system described needs to be able to support the environment in which the fish will live. This means that the water must move through the system in such a way to deliver nutrients and vital elements required the fish to live and remove the waste products they produce. The water interacts with the fish and with the mechanical and chemical

inputs. The water is a transport medium for the requirements of life and must be maintained at the proper oxygen level, temperature, ammonia level, nitrate level, and nitrite level. The control system is responsible for maintaining these parameters. The nutrition or feeding in the system has a large impact on the quality of the water. Since growth is tied directly to the feeding and growth is dependent on water quality the two systems must be balanced and calibrated. The balancing is done by adjusting the feed level to the water quality level. The photoperiod is under control of the system and its length and period is selected by the operator.

System reliability depends on how well the system is designed and on the equipment used. The choice of hardware will depend on the critical nature of the process. If the process can be interrupted by a single fault in a certain device, that device should be made redundant and selected by the quality demanded. This is to say the selection of the equipment should reflect the importance of the device. It is not always the most expensive equipment that can do the best job. These decisions need to be made on an economic analysis of the system.

Detection of faults or errors in the system was done by placing limits or bounds on some of the control parameters. Should a sub-system go out of a control limit the software would switch to a back-up system to take over that process and notify the operator of an off-line device. If the back up should go out of its limits, an alarm or notification of failure would be put up on the screen. By reviewing the data logs the operator could see trends develop, that might indicate an impending failure. An example of this would be reduced pressure in the system as a continuing trend. This

would indicate a pump failure could be expected. The time until total pump failure can only be determined by experience.

The communications that are needed between sub-systems can be determined by examining what interactions are required from one subsystem to another. All sub-systems had a requirement of needing to communicate to a central controller. If the input from one system required an output from another, a link was needed to fulfill this need. This communication was carried out by way of a network that used software to send and receive the data required.

The level of computing requirements for this system could be based on the fastest action that was required to carry out the most critical process. The fastest process in the total system was the control of the main water pump. This required the reading of a flow meter, and the setting of a control register. The overhead in software is what created the maximum time span to carry out the operation. In order to figure out how much time this is, the number of clock cycles used by the controller is counted and added up. This value is divided into the minimum time required to carry out the process. In this system, all processes were of such long time span that the controlling factor was actually the speed required to carry out communications. The speed of the system clock was selected more on a practical decision than on a computed design. Microcontrollers have the ability to operate at extremely high speeds and quite often, the speed selected is based on power consumption or convenience.

The design of computer systems requires that some software be developed to run the system. This software must be developed so that it integrates all functions of the system. When using microcontrollers many new compilers have become available

which allow the use of high level languages so that people with some programming experience can develop this code. This system used a C language compiler (BSO/Tasking 1995) that carried out all of the I/O for the system. This code worked well for the microcontrollers but a higher level of interfacing was required and so a windowed environment was developed to communicate with the system, PC, and operator. This software was important to the systems success because if the interaction of all the systems could not be reported in a simple GUI interface the system would not be a functional piece of hardware.

A communication protocol was adopted from already existing protocols. The two systems used were the standard RS-232 protocol and the control area network (CAN) protocol, both industry standards.

The assembly of the real system should be a straightforward process at this point. Since all the parameters and outputs are known the system should now be a blueprint approach for assembly. The system should be assembled in steps. By doing this any faults can be isolated and eliminated.

CHAPTER 5
MODEL DEVELOPMENT

A model must be developed to create a scheme for the control of the system. The design of a biological model of a species is a complex process due to the many different parameters affecting the living organism. The use of sub-models is required to describe the response of an organism to its environment. A basic model of a fish aquatic ecosystem is depicted in Figure 3.

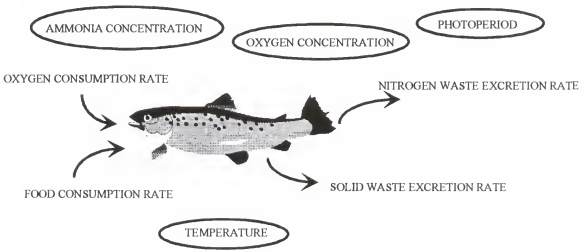


Figure 3. Basic biological model of a fish.

Once a basic over model describing the processes that must be controlled is described, it is broken down into sub-models that are described in detail. Each sub-model consists of its own inputs and outputs but can interact with other sub-models.

Growth Model

The growth of fish has been studied by various people such as Putter, von Bertanlanffy, and Ricker (Allen et al., 1984) and a systematic approach has been developed to determine approximate growth curves for various fish. Fish growth curves tend to follow logistic type of curves (Ricker, 1954). These growth curves are based on many varied parameters that tend to fluctuate from species to species and from the varying environments in which the fish are grown. There is no one model which will fit the entire span of growth for a given fish and environment. This leads to a process that must balance the different stages of growth with a general curve that can be manipulated with the fewest parameters possible to give a reasonable result. The model selected for this study is the von Bertanlanffy growth function (von Bertanlanffy, 1938) which takes the form:

$$W_t = W_\infty * (1 - e^{-K*t})^b \qquad (1)$$

Where

W_t = present weight

W_∞ = final weight

K = growth factor

b = rate of change

t = time

The growth model is then used to predict weight change in the system. This change in weight of the fish is then used to recalculate the amount of feed delivered to the fish. This is a critical step since optimum system performance can be greatly influenced by accurate feeding of the culture. Too little feed and the fish do not grow quickly, too much feed and

there is a degradation of the water quality due to build up of waste product. The growth rate of the fish changes through time as the fish mature. This type of change needs to be adjusted during the life cycle of the product. This system utilizes a growth prediction module that allows the user to compare actual growth to a curve and see how well it fits when the parameters are adjusted. This type of dynamic interaction is required until actual studies can be done which will give the complete growth cycle of the species.

Dissolved Oxygen Model

Life in an aquatic system depends on the presence of oxygen at a certain concentration in the water. Metabolic rates of aquatic animals are very dependent on the proper level of oxygen. The requirement means that the level of oxygen should be maintained at an optimum level through external input. The metabolic rate of fish is usually measured by their rate of respiration, i.e., their rate of oxygen consumption (Froese and Pauly, 1997). The oxygen requirements of fish as used in this model are derived from work done in FishBase (Froese and Pauly, 1997). Although the atmosphere contains approximately 21 percent oxygen gas, it is only slightly soluble in water (Lawson, 1995). Water has the ability to absorb oxygen based on three primary parameters, temperature, salinity, and barometric pressure. Given these three values the saturation concentration of oxygen is calculated using the Standard Methods (APHA, 1989).

Salinity- Although salinity has been defined traditionally as the total solids in water after all carbonates have been converted to oxides, all bromide and iodide have been replaced by chloride, and all organic matter has been oxidized, the new scale used to define salinity is based on the electrical conductivity of seawater relative to a specified

solution of KCL in water. The scale is dimensionless and the traditional dimension of parts per thousand no longer applies.

Temperature in degrees Celsius must be expressed as Kelvin where T is temperature in degrees Kelvin and T_{amb} is ambient temperature.

$$T = T_{amb} + 273.15 \tag{2}$$

Chlorinity- Chlorinity is defined in relation to salinity, Salinity = 1.80655 X chrlorinity. This gives chrlorinity as a function of salinity (S):

$$chl(S) = \frac{S}{1.80655} \tag{3}$$

This then leads to the calculation of oxygen at standard pressure where $C_S(T)$ is concentration of oxygen based on salinity S and temperature T.

$$C_S(T) = \exp\left[-139.3441 + \left(\frac{1.575701 * 10^5}{T}\right) - \left(\frac{6.642308 * 10^7}{T^2}\right) + \left(\frac{1.243800 * 10^{10}}{T^3}\right) + \left(\frac{8.621949 * 10^{11}}{T^4} * -1\right) - chl(S) * \left[3.1929 * 10^{-2} - \left(\frac{1.9428 * 10}{T}\right) + \left(\frac{3.8673 * 10^3}{T^3}\right)\right]\right] \tag{4}$$

Barometric pressure (Bp) is estimated based on elevation above sea level. E is the elevation above sea level in meters.

$$Bp = 10^{\left(2.880814 - \frac{E}{19748.2}\right)} \tag{5}$$

The pressure is then converted to atmospheres by a conversion factor where P is atmospheres

$$P = Bp * 0.0013157895 \tag{6}$$

The oxygen concentration is then corrected for nonstandard atmospheric conditions

$$C_p = C_s(T) * P * \left(\frac{\left(\frac{1 - P_{wv}}{P}\right) * (1 - \Theta * P)}{(1 - P_{wv}) * (1 - \Theta)} \right) \tag{7}$$

Where:

C_p = equilibrium oxygen concentration at nonstandard pressure, $(\frac{mg}{liter})$

$C_S(T)$ = equilibrium oxygen at standard pressure of 1 atm, $(\frac{mg}{liter})$

P = nonstandard pressure, atm

P_{WV} = partial pressure of water vapor, atm, computed from

$$P_{wv} = \exp\left[11.8571 - \left(\frac{3840.7}{T_k}\right) - \left(\frac{216961}{T_k^2}\right)\right] \tag{8}$$

T = must be expressed as temperature degrees K

$$\Theta = 0.000975 - (1.426 * 10^{-5} * T) + (6.436 * 10^{-8} * T^2) \tag{9}$$

T = must be expressed as temperature degrees C

A typical curve relating temperature, salinity, and pressure is shown in Figure 4. This curve is representative of the internal calculations that are made by the PC.

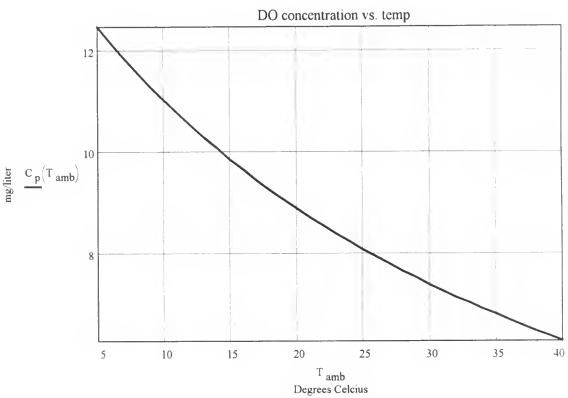


Figure 4. Dissolved oxygen as a function of temperature.

The above model is used to determine the maximum oxygen concentration in water for given conditions. This sets the upper bound for which the water will be oxygenated. Once this level is obtained there is no need increase the pumping level of air as the extra effort is not needed or desired. If there is a need to increase the oxygen level, actions can be taken such as reducing the temperature. This may be needed if there is a reduction in the ability to produce air and since a reduction in temperature would reduce the metabolic rate of the culture it would also decrease the consumption of oxygen in the system. These

measures are not good production practices but serve to correct problems that might occur in an emergency situation. Although the transfer of oxygen to water can be calculated and the consumption rate of the culture can be calculated, the algorithm for the control of oxygen in the water is based on a feedback method from the sensor and has little to do with the overall calculation of the many dynamic variables in the system.

The controlling algorithm for the air pump in the system is based on a set point that is input by the system database or the user. This set point is compared against the measured value from the DO meter and subtracted. The value of the subtraction is then compared against the error range and if it lies inside this band nothing is done. Should the value fall outside, it would mean an corrective action is needed. If the value is positive, it means the air needs to be increased and the pump speed is increased, if the value is negative the speed would need to be reduced. The amount by which the value is reduced is a percentage of the error that has occurred. This is a proportional value and a proportional constant must be determined that will be used such that there will not be an overshoot of the target value. In this way there is a smooth transition to inside the error range. Other considerations are the time it takes to affect the increase in the level of oxygen. The diffusion of oxygen into water is not a very efficient process and the changing of its value takes from minutes to hours. With this in mind, the time between checking and acting

upon the system needs to be long enough to account for some measurable change, this is the time constant. There was about a one minute time constant used in this process

Photoperiod Model

Photoperiod can influence the growth and development of fish in various ways. The length of period that the fish are in light controls their activity. This activity includes feeding, spawning, and metabolic activity. In a system that is located in an enclosure, the photoperiod must be controlled by external means. This system uses the location desired in terms of latitude, longitude, and time of year to determine the length of time that will be used as photo-active, or when the lights are on. The system uses only an on/off system of lighting. In a more complex system, it is desirable to use a gradual system of turning on and off the light as is found in nature. This model has the ability to carry out this function but it is not used in this study.

Solar radiation impinges on the earth and both lights and heats the surface. This process is controlled by the rotation of the earth, the motion of the earth around the sun, and the tilt of the earth on its axis. These conditions bring about different photoperiods depending on the latitude, longitudes and time of year. The sun's position must be located in the sky, which will give the incident angle that the ray of sunlight will strike the earth. All calculations involving solar radiation are based on solar time. Solar time is described as the angular motion of the sun in the sky. This time must be converted from Local Standard Time (LST) with corrections to give apparent solar time (AST). This is found by the following:

$$AST = LST + ET + 4*(LSM-LON) \quad (10)$$

Where ET = Equation of time, minutes

LST = Local Standard time

LSM = Local Standard Time Meridian, degrees

LON = Local Longitude, degree

4 = minutes of time required for one degree of rotation of the earth

Where

$$ET(n) = 9.87 * \sin\left(4 * \frac{n-81}{364}\right) - 7.53 * \cos\left(2 * \pi * \frac{n-81}{364}\right) - 1.5 * \sin\left(2 * \pi * \frac{n-81}{364}\right) \quad (11)$$

With n = day of year (1-365).

The procedure is then to use the day of the year to correct with the equation of time and find AST based on location longitude (LON) and LST that is the present time from the equation (10).

The hour angle is calculated by the equation:

$$h = (AST - 12 * hr) * (15 * {}^{\deg}/_{hr}) \quad (12)$$

The declination is then calculated by the equation:

$$\delta = 23.45 * \deg * \sin(360 * {}^{284 + n}/_{365} * \deg) \quad (13)$$

Then calculate the solar altitude by the equation:

$$\alpha = a\sin(\cos(L) * \cos(\delta) * \cos(h) + \sin(L) * \sin(\delta)) \quad (14)$$

With L = latitude

The solar azimuth could be calculated also so that an incident angle of the sun could be used to calculate intensity of the sun. It is not done in this case because only sunrise and

sunset are required. Once the altitude of the sun is known it can be determined if it daylight or dark. This is used to then tell the system to turn on the lights or to turn them off.

Waste Management Model

The quality of the water is determined by many parameters that can be controlled by different methods. The waste management model is set up to control the build up of solids in the water and the build up of nitrates or nitrites in the water. The system used manual input from the operator that was then compared to the desired values and an adjustment was made as to the amount of water that was to be wasted. The wasting of water is the removal of water from the system.

The control of waste in an aquaculture system is done primarily through the removal of the waste and exchange of water. Most systems have some form of solids removal. This system uses a settling filter to allow large solids to accumulate in the bottom of a cone that is then periodically drained and removing solids. This also accomplishes the exchange of water need to reduce the build up of nitrates in the system. The exchange rate is the percentage of water that is replaced each day in the system. The following example is from the system in this study where

L_t= length of tank

H_t= height of tank

W_t= width of tank

r_f= radius of filter

h_f= height of filter

N_{tanks}= number of tanks

Vol_{tanks}= volume of tanks

Filter_Vol= volume of the filter

Vol_{tot}=total volume of the system

ER%= daily percent exchange rate

Per_cent_Waste=amount of water removed daily

$L_t = 22$ (in) $H_t = 10$ (in) $W_t = 12$ (in) $r_f = 3$ (in) $h_f = 36$ (in)

$N_{tanks} = 4$

$$Vol_{tanks} = L_t * H_t * W_t \quad (15)$$

$$Filter_Vol = \pi * r_f^2 * h_f \quad (16)$$

$$Vol_{tot} = N_{tanks} * Vol_{tanks} + Filter_Vol \quad (17)$$

$Vol_{tot} = 50.121$ gal

ER% = .04

$$Per_cent_Waste = Vol_{tot} * ER\% \quad (18)$$

Per_cent_Waste = 2.005 gal

In this example there is a 4 percent daily exchange rate resulting in approximately 2 gallons of water exhange each day.

Time required to run the pump is based on the flow rate of the pump and the time that the pump is run is based on an equal distribution throughout the day. There is a minimum requirement that the pump be turned on at least for 10 seconds. There was a minimum time span of one second in the controller. This means that no event could occur at a quicker response time. The PC used a one-minute time span for testing of wasting events.

Time_per_pump = 10 (sec)

$Pump_time_{tot}$ = 120.20 (sec)

The number of times of pumping must be an integer so that a C language function is used to convert to the integer required

$$Times_Pump_day = floor(Pump_time_{tot}/Times_per_pump) \quad (19)$$

Time_Pump_day = 12

There is the option of manually inputting the time between exchanges and inputting the time the pump is turned on. When either of these selections is done, the algorithm then adjusts the other parameter.

Knowing the number of times per day that the system would be used it was then necessary to calculate the time at which the pumping would occur

$$Time_Between_pumps = 24hr/Times_Pump_day \quad (20)$$

Time_Between_pumps = 120 min

This gives the interval that then starts adding it to 12 AM and sequentially turns the pump on at every interval. Upon time out of the timer, a message was sent to the controller. This message included the command to turn the waste pump on and the amount of time that the pump was to be turned on. Once the PC sent the message, the process was under control of the microcontroller system, which then acted to carry out the commands independently of the PC. Figure 5 shows a flowchart of the sequence of events that the waste routine entails.

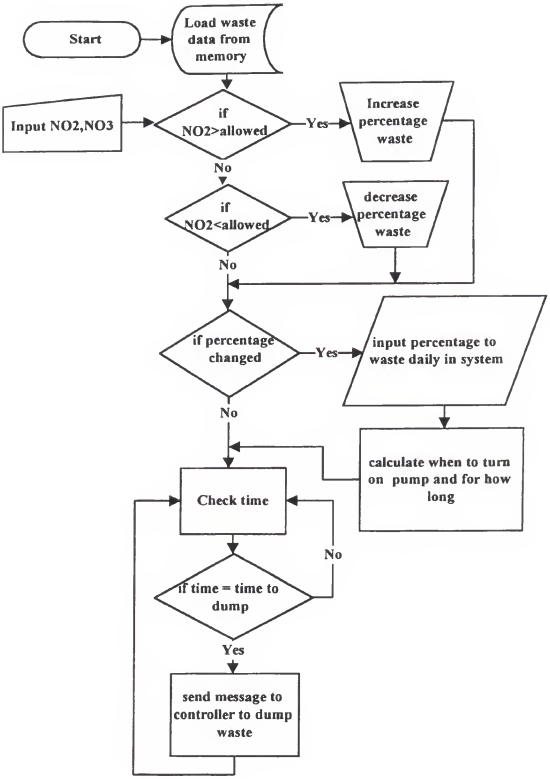


Figure 5. Waste model flowchart.

Flow Control

Some of the most important controlling factors in an aquaculture system are the flow rate of oxygen, water, and feed through the system. The sizing of all components will determine the flow rate or the flow rate can determine the size of the system components. Aquacultural systems are dynamic systems always in states of change. The best way to manage these systems is with the ability to vary the flow rate of water to match the parameters of the system as they change through time. Varying the flow through the system can also help maintain optimum quality of the water that should lead to better growth.

Many of the systems parameters are dependent on the period of time each has to react in the system. A good example of this is the filters ability to reduce the levels of ammonia and nitrites in the system. There is an optimum period of reaction time for the bacteria to consume the ammonia and convert to the final product of nitrate. This time is based on the filter size and amount of ammonia present and the flow rate through the filter. As the fish grow the amount of feed is increased to match their requirements. This increase is typically accounted for by designing the filter large enough to match the final size of the culture. So for most of its life cycle, it is over designed to meet this requirement. With a variable flow rate, the filter can be designed for maximum flow rate. There is often a minimum flow rate required for such things as fluidization or submergence of the filter.

Flow of feed into the system is critical in maintaining the water quality. This flow is controlled from a different method that that of water flow. The feed is distributed into the

system at one time period with the amount delivered determined by how long the motor is turned on. The feed is only delivered once a day to keep the system simple. A more detailed description of the feeding process can be found in the growth and feeding section of the paper.

The flow of oxygen into the system was controlled in a similar manner as flow of water, that is, by use of a variable pump to change the flow rate. A more detailed description of how the process was done is described in the oxygen model.

The ability to change the flow rate can also be used as a safety measure in the system. When an out of bounds event occurs it can be compensated for by changing the flow rate. Oxygen can be increased or decreased to meet metabolism changes that occur from feeding or temperature changes. Feeding levels can be changed to meet growth or decreased because of other environmental changes due to failures in sub components. Flow of water through the system can be altered to match growth or other environmental changes that might affect filter performance.

Typically, the flows through most aquaculture systems are set at fixed rates with the exception of feed that is mainly done by manual control. The use of a microcontrolled system allows for the automation of control of all flows through the system. Most microcontrollers have the ability to generate a variable signal that can either be regulated by feedback, or be set, or be based on an algorithm. The signal comes in two forms, pulse width modulation (PWM) or digital output. Simple digital output is the switching on or off a subsystem, such as a feeder. PWM is a more complex operation consisting of two basic components, frequency and duty cycle. The frequency of the signal is the time from on to on again or the time that elapses between each turning on of the device. The duty cycle is

the time from on to off or the time that the device is turned on. The PWM cycle can be thought of as the percent of time that the device is turned on and the frequency of turning the device on. This type of signal will give an effective voltage that is lower that the actual peak voltage. The signal when filtered should be sufficient to be applied to the winding of a motor. Quite often, the winding of the motor will act as a low pass filter. The output generated by a low duty cycle is shown in Figure 6. The low duty cycle effectively lowers the voltage that the motor sees and the motor runs at a slower speed. In Figure 7 the output has a high duty cycle, because for the time the voltage is on it is effectively a higher voltage and the motor will run at a higher speed. Since the actual voltage applied is still at the maximum level, the torque induced into the motor is much greater than the level that would be if the actual voltage were reduced. This results in a cooler running motor with lower power consumption.

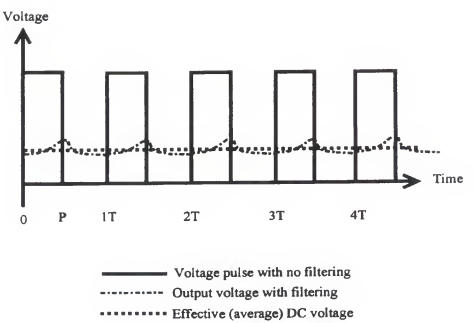


Figure 6. PWM output with a low duty cycle.

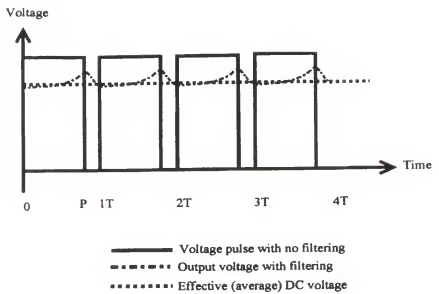


Figure 7. PWM output with a high duty cycle.

It is in this manner that a DC motors speed can be controlled. This is much more effective than actually reducing the voltage to the motor as that scheme will cause the motor to draw more current and overheat. The cost of variable voltage devices makes them unattractive. PWM delivers the same peak voltage each cycle and the torque delivered is much higher. This allows for much better control of the speed of the motor as well as a better resolution in the bandwidth of the desired speed.

In the system developed in this study, a flow meter was used to measure the flow of the water through the system. This flow meter is a paddle-wheel type insertion flow sensor that utilizes precise digital technology to accurately monitor flow rates from 1 to 20 feet per second. The TTL/CMOS compatible square wave pulse output was isolated optically and then converted to a TTL signal. This pulse was counted using a port on the

microcontroller described as a timer/counter port. The port senses a change of signal and then increments a counter. The counter is then compared against a time base and a pulses per second reading is derived. This reading is then converted to a flow rate of gallons per second. Once the flow rate is determined, the PWM output to the pump motor is adjusted so that the flow will be changed to meet the required set point. The first attempt to control the motor was done by selecting a set point and comparing it against the flow rate measured. If the flow was higher than the desired set point, the PWM value was reduced by a constant value. If it was lower, then the set point the PWM value was increased by a constant value. This also worked except that the system took a long time to stabilize. Next, the constant value of change was made based on the difference from the set point. This worked as well but the system then oscillated quite a bit and never settled down. The third method tried was to use proportional control. This method found the amount of error from the set point and then used a proportion of this error to correct the PWM value. If the error was negative, it decreased the value, if the value was positive then it increased the value. After a few attempts at finding the correct proportional value to use the system was then very stable and quickly reached this rate.

The software functions to solve other flow problems that occurred with the pump. These problems had to do with the timing of adjustment the motor speed. The use of proportional control meant there would be a greater discrepancy in the flow rate, as a large change is made to the speed, should there only be a small difference then only small changes are made. When the pump was first started there was a need to put a minimum amount of torque on the motor so that it could overcome the inertia of the pump. This was also accounted for in software. The lowest setting on the PWM was determined and

whenever the pump was started, this value was used to start the pump. When the flow meter quit, it was usually due to an obstruction in the vanes rather than pump failure. Once the flow meter stopped, the system would then revert to a secondary scheme for control. The system would learn what the appropriate level of control for the motor was and set it at that value. This was done by logging the values of the differential pressure gauge. When the flow meter stopped the last value read would be used to control the system. The flow meter would then read zero. This would show up on the system status screen. In order to confirm if the flow meter was stuck, a pressure gauge was put in the system to read the level of pressure at the pump outlet. If this showed an acceptable level then the pump was considered as functioning and a secondary level of control was initiated. If both the flow meter and the pressure gauge read zero a pump failure message was sent to the monitor screen to notify the operator of pump failure. There are other conditions which must be accounted for in the system, such as loss of water, clogged filter, air entrapment in the lines, and so forth. These problems can be analyzed and solved through a mini expert system in software as shown in Figure 8.

One significant problem with controlling the flow through a system is the accumulation of waste material in the systems flow channels. This causes many hydraulic problems that impede the flow and cause restrictions through the system. The changing of pump speed can compensate for some of these restrictions but quite often, there is a need to clear out the restrictions. The removal of waste from the system as quickly as possible helps to reduces this fouling problem. The use of a microcontrolled waste removal system facilitated this solution. With the use of sensors to detect restrictions in flow, the system

can be turned on to dump the unwanted waste. Some of this logic was not implemented due to lack of time and resources.

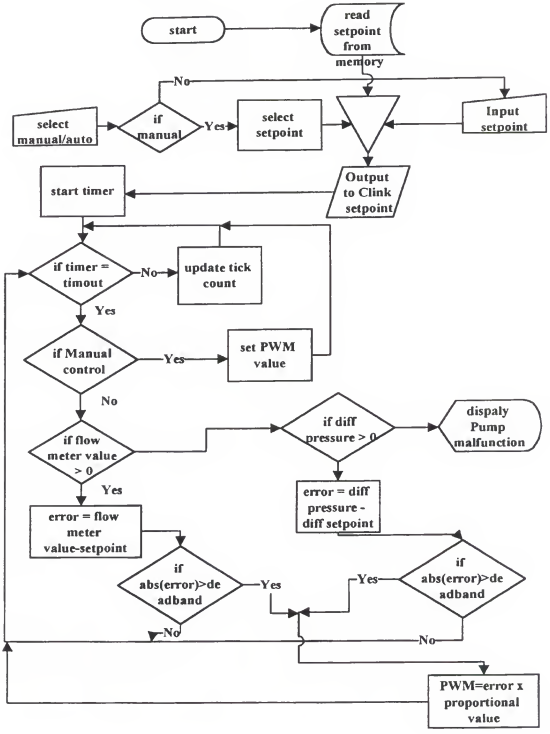


Figure 8. Flow model flowchart

CHAPTER 6
COMPUTER CONTROL SYSTEMS

Computer control systems have matured into a well-defined technology with constant improvement in methodology and adaptation. The use of computers in the agricultural field has been mainly confined to planning and development. In industry, computer control systems are readily available for all aspects of systems control. The main difference in industrial and agricultural control is the added aspect of biological impact causing a steady state system to become a non-stationary dynamic system. The managing of biological systems requires that control systems adapt to the changing environment which can have many unforeseen inputs and changing variables. Typically today most agricultural systems are managed by manual labor with local knowledge comprising the decision making process. It will be difficult to outperform the local knowledge input from a farmer and the systems must learn the processes through database diagnostics.

Processes are very adaptable to computer control because every process involves three kinds of flow: flow of materials, flow of energy, and flow of information (Savas, 1965). Information is the key to any process control. Information flows are both directed to the system and to the operator. This information is in the form of direct feedback measurements from sensors, process update information, energy usage, equipment status, alarm conditions, and so forth. The computer is an ideal management tool for information

processing. There are two basic requirements of a control computer system and it associated equipment:

1. The ability to communicate effectively and efficiently with the process instrumentation and control equipment.
2. The ability to operate upon data rapidly enough to provide effective control action (Savas, 1965).

These two parameters will dictate how the rest of the system is put together. The two key words here are information and timing.

The use of *soft logic* (software logic) is one of the main components in the development of integrated control computer systems. Soft logic packages are actually software which is intended to be used as the application program for the machine or process (Carrow, 1998). These programs are the heart of any control system and can enhance the performance of a system when written in such a way that anyone in the particular industry can understand them. Programming languages have developed to a point where the complex task of writing code has been reduced to graphical concepts which relay information to even untrained users. The development time for producing code has been reduced but at the price of increased use of memory. The increased memory demand has been offset by the reduction of cost for that memory.

There are three types of computer process control strategies that are used, off-line, in-line and on-line. The first two involve human interface for control and are not sufficient to handle the dynamic requirements of an intensive-changing environment. The on-line system requires that the computer maintain the system as the primary controller. On-line systems are further broken down into open loop and closed loop systems. In an open-loop

system, a human operator is used in the process to interject necessary information to keep the system stable. The goal is to obtain a closed-loop system where by the computer maintains all the control functions necessary to make the system operate. Often a system needs to be started as an open-loop system until the computer adapts to the system and is then able to take over the process control.

Distributive computer control systems are ones in which the process is broken into various sub-systems and each sub-system is able to control its own process. These systems allow the simplification of decision process so that a closed-loop sub-system is achieved. Where a single computer control system needs to time-share the operations in its control logic, a distributive system can run parallel processes and enhance the performance of the overall system. Scheduling processes with distributive systems allows for a faster response time and more accurate control of the process. Time-sharing is often a bottleneck in computer control systems that can be overcome with distributive systems. In Figure 9 the differences in time required for processing is compared for the various systems.

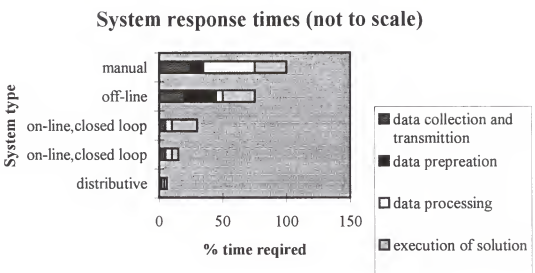


Figure 9. System response times.

Until recently, automated control for the physical process of systems used analog systems that were more of an art than a science. Computers have created a new concept in the control field by digitizing the process that allows discrete decisions to be made. The analog side of the equation is still there but is converted to digital signals by conversion. The decision process for systems today is more complicated and requires a faster response time to keep the systems functioning. The improvement in control systems with the use of computers has lead to standardization of procedures and control. This has not however removed the human operator from the system. Regulation of a process is the primary consideration of control theory. A complex system requires that many different variables be regulated at the same time in order to maintain system stability. Dynamic non-stationary processes require that the system adjust continuously. This type of process needs to have information quickly and decisions must be made quickly. Small independent computers are ideal for this type of problem. The need to transfer information quickly between independent systems has led to the development of small communications networks that are capable of large data rates in short periods of time. This creates the ability for the devices to act independently and yet be centrally controlled by more powerful computers that can analyze data from all aspects of the system, thus tying together the complete system into a functioning unit. Since one unit can monitor all devices it can collate the information and interact with the human operator who may have the ultimate experience and judgment for the system. The central computer can have more complex algorithms that decide when interactions of processes need to be done to keep the system in balance.

The hardware to operate computer control systems is presently well developed and in use throughout industry. The ability to interface and interact with the hardware is the

job of the program language and programmers. Programs that use good soft logic are the best at accomplishing this task. Programming is done at two basic levels, machine to machine interface and human to computer interface. The combining of machine to machine logic requires that sensors and inputs from devices be understood by the computer and acted upon. This often done without any human intervention other than the proper logic that is installed by the program. This logic must be understood by the programmer who must know what the input parameters will mean and reason accordingly with the proper output logic. An example of this is the control of temperature. The input is often a single value that gives a reference point of how the process is doing. The logic must make decisions based on this input and understand the consequences of the action that is devised. In the case of temperature, the decision must be whether to increase or decrease the heat flow into the system and for how long. Another periodic reading may give a trend as to the state of the condition and be used to decide further actions. When the operator is not involved in the decision making process some parameters and limits must be set by the overall system control as a guideline for the computer logic. If the system is being monitored by an operator, the final control decision can be made by that operator. If the decision must be made quickly, it is often best to set limits and let the computer do the control. Human machine interfacing is the interaction of the computer with the operator. This process must be ergonomically correct in order to achieve optimum results from the interaction. Computer screens must be displayed in such a way that they can be easily understood and interacted with. To accomplish the task there can not be too much information on one screen. There is a tendency for human operators to quickly be overwhelmed with large amounts of data. It is best to break down the screen into smaller

screens that display specific information about certain aspects of the process. All input data should be scanned for "reasonableness". This means that whenever an input is entered it should be compared to known values that are acceptable to the system process, if the value is out of range the operator should then be queried for his desire to input the value. It is best to color code sections of the display to indicate similarities of process control. Color can also be used to show alarm conditions, incorrect input, or change of values in the system.

Networking Microcontrollers

One of the latest innovations in the development of microcontroller design is the ability to network microcontrollers by way of on board peripherals. This allows communication between devices as well as to a central unit that can monitor the system of controllers. This ability to communicate between controllers is not new but the reduced hardware and increased reliability is. There is also a drive to establish a standard for communicating between devices so many different types of controllers can be on the same network.

CAN 2.0A Protocol Basics

The FP-CAN Protocol is a communication protocol built on top of the CAN 2.0A protocol. This protocol was developed at the University of Florida by F. Fialho and P. Fowler for use in this system. CAN 2.0A specifies that a message is composed of the following fields as shown in Table 3. These fields are divided by both byte and bit manipulation. This allows for easy deciphering of the code when it is read. As shown in Table 5 the CAN 2.0A protocol defines the following parts of a message:

ID 11-bit message identifier, used for arbitration; must be between 0 and 2031.

RTR Remote transmit request; 1 = remote frame; 0 = data frame.

DLC Data length code; must be a value between 0 and 8.

Data Optional data bytes; the number of actual data bytes depends on DLC.

Table 5. CAN 2.0A protocol.

Byte	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
1	ID.2	ID.1	ID.0	RTR	DLC.3	DLC.2	DLC.1	DLC.0
2	Data1.7	Data1.6	Data1.5	Data1.4	Data1.3	Data1.2	Data1.1	Data1.0
3	Data2.7	Data2.6	Data2.5	Data2.4	Data2.3	Data2.2	Data2.1	Data2.0
4	Data3.7	Data3.6	Data3.5	Data3.4	Data3.3	Data3.2	Data3.1	Data3.0
5	Data4.7	Data4.6	Data4.5	Data4.4	Data4.3	Data4.2	Data4.1	Data4.0
6	Data5.7	Data5.6	Data5.5	Data5.4	Data5.3	Data5.2	Data5.1	Data5.0
7	Data6.7	Data6.6	Data6.5	Data6.4	Data6.3	Data6.2	Data6.1	Data6.0
8	Data7.7	Data7.6	Data7.5	Data7.4	Data7.3	Data7.2	Data7.1	Data7.0
9	Data8.7	Data8.6	Data8.5	Data8.4	Data8.3	Data8.2	Data8.1	Data8.0

FP-CAN Protocol Basics

The FP-CAN protocol specifies some communication standards for a CAN network. This protocol was created to extend the functionality of the microcontroller design that was developed. The complexity of this multiple type of network was simplified by use of the CAN system for communications between devices. This is because all arbitration was done in hardware therefore eliminating software development in this system.

1. Four logical networks (0…3) can coexist on the same physical network. Each device belongs to one and only one logical network; each logical network consists of one "head" (the device controlling the logical network) and up to 255 "feet" (devices controlled by the head). CAN protocol restrictions, logical network 3 is limited to 239 feet; a maximum of 1008 devices can be connected to the physical network.

2. Messages within a logical network are either from the head to the feet or from a foot to the head. Heads have address 0 on the logical network. A foot may have an address ranging from 1 to 255. Broadcast messages may be sent, by using device address 0 as the destination and these messages will be received by all feet on the logical network.
3. The ID field is divided into 3 sub-fields as shown in Table 6.

Table 6. ID field breakdown.

DIR (ID 10)	Direction of the message (0 = Head to Feet; 1 = Foot to Head).
NET (ID.9..8)	Logical network of the message (0…3).
AD (ID.7..0)	Address of the foot sending or receiving the message (1…255 or 1…239), or 0 for a broadcast message.

4. The RTR field is currently not used, and should be set to 0.
5. Data byte 1 contains the message number (1…254); messages number 0 and 255 are reserved for future use. The remaining data bytes (2…8) may contain additional information or data.
6. Data is transferred between head and feet using data banks. A total of 32 data banks (0…31) are allowed. Each data bank is 48 bytes long and is divided into 8 pages (0…7) of 6 bytes each. Each data transfer message transmits the page address (8 * bank + page) on data byte 2 and a page of data on data bytes 3 to 8.
7. Although not recommended, it is acceptable for a device in one logical network to send a message to a device in another logical network.

FP-CAN Protocol Messages

The message number is contained in the first data byte. The following FP-CAN messages are currently defined:

1. Wake up (head to foot): tells the foot to listen and respond to messages. It cancels the effect of message 2 (sleep).
2. Sleep (head to foot): tells the foot to stop responding to messages. Subsequent messages will be ignored until the foot receives message 1 (wake up).
3. Identify yourself (head to foot): requests a response from the foot stating what kind of device it is. The foot should respond with message 4 (here I am).
4. Here I am (foot to head): tells the head what type of device the foot is, in response to message 3 (identify yourself). The device type (1...254) is contained in data byte 2; device types 0 and 255 are reserved for future use.
5. Send data (head to foot): requests data from a foot. Data byte 2 contains the data bank containing the data (0...31) and data byte 3 contains a mask of which pages in the bank should be sent. The foot should respond by sending one message 6 (here is the data) for each bit set in the page mask.
6. Here is the data (foot to head): sends the requested data to the head, in response to message 5 (send data). Data byte 2 contains the page address (8 * bank + page) and data bytes 3 to 8 contain the data.
7. Set data (head to foot): sends data to a foot. Data byte 2 contains the page address (8 * bank + page) and data bytes 3 to 8 contain the data. The foot should accept and register the data, and respond by sending message 8 (data was set).
8. Data was set (foot to head): tells the head that the data was received, in response to message 7 (set data). Data byte 2 contains the page address (8 * bank + page).

FP-MOM CAN Configuration

FP-MOM is a motherboard that uses the Philips 80C592 microcontroller. This board was developed and designed by F. Fialho and P. Fowler. The CAN controller registers have to be properly configured. This configuration is done on the bit level with each bit representing a certain control aspect within the system. It is important to understand the use of each bit as control of the system depends on the correct settings.

Byte 0 (CR): Control register.

Bit 0.7 (CR.TM): Test mode; should always be 0.

Bit 0.6 (CR.S*): Sync; should be 1 for 2-edge synchronization.

Bit 0.5 (CR.RA*): Reference active; should be 1 for ½ AV_{DD} reference output on REF.

Bit 0.4 (CR.OIE): Overrun interrupt enable; should be 0 to disable overrun interrupts.

Bit 0.3 (CR.EIE): Error interrupt enable; should be 0 to disable error interrupts.

Bit 0.2 (CR.TIE): Transmit interrupt enable; should be 0 to disable transmit interrupts.

Bit 0.1 (CR.RIE): Receive interrupt enable; should be 0 to disable receive interrupts.

Bit 0.0 (CR.RR): Reset request; 1 = reset the CAN; 0 = normal operation.

* can be modified only if CR.RR is 1.

Byte 1 (CMR): Command register.

Bit 1.7 (CMR.RX0A*): RX0 active; should be 1, for RX0 = CRX0.

Bit 1.6 (CMR.RX1A*): RX1 active; should be 1, for RX1 = CRX1.

Bit 1.5 (CMR.WUM): Wake-up mode; should be 0 for differential wake-up.

Bit 1.4 (CMR.SLP): Sleep; should be 0, for normal operation.

Bit 1.3 (CMR.COS): Clear overrun status; 1 = clear overrun; 0 = nothing.

Bit 1.2 (CMR.RRB): Release receive buffer; 1 = release buffer; 0 = nothing.

Bit 1.1 (CMR.AT): Abort transmission; 1 = abort pending transmission; 0 = nothing.

Bit 1.0 (CMR.TR): Transmission request; 1 = transmit message; 0 = nothing.

* should be modified only if CR.RR is 1.

Byte 2 (SR): Status register (read only).

Bit 2.7 (SR.BS): Bus status; 1 = bus-off; 0 = normal.

Bit 2.6 (SR.ES): Error status; 1 = error count high; 0 = normal.

Bit 2.5 (SR.TS): Transmit status; 1 = transmitting; 0 = idle.

Bit 2.4 (SR.RS): Receive status; 1 = receiving; 0 = idle.

Bit 2.3 (SR.TCS): Transmission complete status; 1 = complete; 0 = incomplete.

Bit 2.2 (SR.TBS): Transmit buffer access; 1 = free; 0 = busy.

Bit 2.1 (SR.DO): Data overrun; 1 = overrun; 0 = normal.

Bit 2.0 (SR.RBS): Receive buffer status; 1 = message; 0 = idle.

Byte 3 (IR): Interrupt register (read only; reset to 0x00 when read).

Bit 3.4 (IR.WUI): Wake-up interrupt; 1 = left sleep mode; 0 = normal.

Bit 3.3 (IR.OI): Overrun interrupt; 1 = data overrun; 0 = normal.

Bit 3.2 (IR.EI): Error interrupt; 1 = error limit exceeded or bus off; 0 = normal.

Bit 3.1 (IR.TI): Transmit interrupt; 1 = transmission completed; 0 = normal.

Bit 3.0 (IR.RI): Receive interrupt; 1 = message available; 0 = normal.

Byte 4 (ACR): Acceptance code register; should be 0x80*DIR + 0x20*NET; DIR is 1 if the device is a head and 0 if the device is a foot; NET is the logical network number (0…3).

Byte 5 (AMR): Acceptance mask register; should be 0x1F.

Byte 6 (BTR0): Bus timing register 0; should be 0x40*SJW + BRP.

Byte 7 (BTR1): Bus timing register 1; should be 0x80*SAM + 0x10*TSEG2 + TSEG1.

Table 6 specifies values for BTR0 and BTR1 for different baud rates and two oscillator frequencies:

Table 7. Bit settings for baud rate based on crystal frequency.

baud	clock	SJW	BRP	SAM	TSEG2	TSEG1
4800	11.0592	3	47	1	6	15
4800	14.7456	3	63	1	6	15
6400	11.0592	3	47	1	4	11
6400	14.7456	3	63	1	4	11
7200	11.0592	3	47	1	3	10
7200	14.7456	3	63	1	3	10
9600	11.0592	3	23	1	6	15
9600	14.7456	3	31	1	6	15
12800	11.0592	3	23	1	4	11
12800	14.7456	3	31	1	4	11
14400	11.0592	3	23	1	3	10
14400	14.7456	3	31	1	3	10
19200	11.0592	3	11	1	6	15
19200	14.7456	3	15	1	6	15
25600	11.0592	2	11	1	4	11
25600	14.7456	2	15	1	4	11
28800	11.0592	2	11	1	3	10
28800	14.7456	2	15	1	3	10
38400	11.0592	2	5	1	6	15
38400	14.7456	2	7	1	6	15
51200	11.0592	1	5	0	4	11
51200	14.7456	1	7	0	4	11
57600	11.0592	1	5	0	3	10
57600	14.7456	1	7	0	3	10
76800	11.0592	1	2	0	6	15
76800	14.7456	1	3	0	6	15
102400	11.0592	1	2	0	4	11
102400	14.7456	1	3	0	4	11
115200	11.0592	1	2	0	3	10
115200	14.7456	1	3	0	3	10
153600	11.0592	0	2	0	2	7
153600	14.7456	0	3	0	2	7
204800	11.0592	0	2	0	1	5

Byte 8 (OCR): Output control register; should be 0x1A.

Byte 9 (TR): Test register; do not change.

Bytes 10...19: Transmit buffer.

Bytes 20...29: Receive buffer.

General

The Flavio-Phil (FP) system is based on a motherboard-daughterboard concept, in which one or more daughterboards are connected to a motherboard. This entire concept and development was done at the University of Florida by F. Fialho and P. Fowler. The connection is made through a set of headers whose positions on the board are set by the FP standard given here. The pins on the headers are also defined by the standard, such that any daughterboard should be able to connect to any motherboard. The current standard defines two headers. The first one is for the memory connections (address, data and control lines) and for the raw CAN connections (connecting directly to the microcontroller). Each of the two headers can have 34, 40 or 50 pins. The pinout for pins 1-34 is the same for all three header sizes and the pinout for pins 35-40 is the same for the 40 and 50 pin headers. Pin 1 of the memory header is aligned with pin 7 of the ports header (300 mil / 7.62 mm offset). The distance between each line of pins in the memory header and the corresponding line of pins in the ports header is 3000 mil (76.2 mm). The layout of the headers is shown in Appendix E as header layout. The memory and ports headers pinout are currently defined as shown in Appendix E. This motherboard contains a regulated power supply, CAN interface, an 8-pin DIP switch for selecting the board address, 64 KB of EPROM and 32 KB of RAM. The EPROM contains the code addresses 0x0000-0xFFFF, and the RAM contains the data addresses 0x0000-0x7FFF.

The addresses are demultiplexed on the motherboard using a 74HC573 latch. Data addresses 0x8000-0xFFFF are available for use on the daughterboards.

CLINK Daughterboard

The Computer-LINK (CLINK) daughterboard is designed to serve as a data logger and a communication link with a PC. It collects data from the other units on the network and stores it in memory. It must be programmed such that power interruptions will not cause logged data to be corrupt. It also is designed to use a Real Time Clock (Dallas DS1387, DS1287 or DS12887) with some amount of battery-backed RAM for storing parameters which need to be changed frequently. A Reset button is also present, as well as 4 other buttons, one switch and a 14-pin header to interface with a LCD. The pinout on the 14-pin header is shown in Appendix E as LCD pinout. Note that the pinout is inverted, to enable the LCD to be mounted on the wall of a box. Three LEDs are on the board: one green (power), one yellow (communication) and one red (error). These LEDs were useful when debugging software. A 6-position RJ-11 contains connections for both CAN and power. The connections are shown in Appendix E (viewed from above) as RJ-11 Connector Cable. Pins 1 and 6 are connected to pins 1 and 2 of a terminal block. A pair of wires should connect this terminal block to the power terminal block on the motherboard (the polarity is irrelevant for the CAN-592 Motherboard). Pins 2 and 5 are connected to pins 3 and 4 on the Ports header. The Memory header has 34 pins and the Ports header has 40 pins, as in the CAN-592 Motherboard. The connections on the headers are shown in Appendix E as CLINK header pinout.

AGUA Daughterboard

The AGUA daughterboard is designed for aquaculture applications. The connections on the Ports header are shown in Appendix E as AGUA header pinout.

CHAPTER 7
MICROCONTROLLER HARDWARE DESIGN

Design of microcontroller systems in agriculture is a complex task involving knowledge in several fields of expertise that includes electrical engineering, mechanical engineering, agricultural engineering, software engineering, biology and physics. There is a procedural method for developing such systems. The method is very similar to the process outlined in this dissertation. The steps are as follows:

1. Describe the objective of the system.
2. List all required inputs and outputs from the system.
3. Decide on the minimum speed required to carry out the quickest process in the system.
4. Decide what the power requirements for the system will be.
5. Decide memory requirements.
6. Decide on communication requirements.
7. Select a microcontroller family, (4-bit, 8-bit, 16-bit) that matches the previous selections.
8. Select the proper software for program development.
9. Draw a schematic of the circuit of the system.
10. Have a printed circuit board designed and made.
11. Test the board.
12. Connect all peripherals and test system.

The design of a circuit board first starts with the basic concept that one desires to produce. This concept must then be expanded to include all interfacing devices that will be required to make the system functional. These devices include sensors, input/output, communications, status indicators, and memory. With this compellation of devices a microcontroller chip must be selected that will accommodate them. There are many decisions that must be made when selecting the proper chip. The first choice in architecture is the word size of the chip. In general, a word is the unit that the processor uses when data is to be operated on or instructions are to be performed. A word is a fixed group of bits that varies according to the controller; it can be 4-bit, 8-bit, 16-bit, or 32-bit in length. The length of the word will determine the instruction set, the ability to address memory, and speed of the controller. The next consideration in microcontroller selection is the speed at which the chip will run. This is given in MIPS. Selecting the correct speed is important in real-time operating systems when certain processes must occur in a timely fashion. It must be determined beforehand the largest increment of time that can elapse before a vital process must happen. This then becomes the basis for the speed of the microcontroller. The power requirements of the system should then be decided. Often it will be some external component that will be the maximum voltage required for the system. Microcontroller chips come in several different voltage ranges. Generally the lower the voltage the lower the power consumption of the system. This can be critical if the system is to run off batteries or there is a need to maintain a low heat output from the system. The amount of memory required to carry out the process should be considered at this point. If the process is simple, a controller with on board memory may be adequate. Microcontrollers can be interfaced with external memory that can give enough space to do

complex algorithms, keep track of data, and do logical control. When external memory is used, the system can be updated by exchanging memory chips without the need to replace the entire system. Next, the number and type of input/output (I/O) ports that are available should be decided on. The number of ports required is based on how many peripherals are needed to make the system work. The type of ports available are digital, analog to digital (A/D), timers, counters, pulse width modulated, and memory. Microcontrollers come with various numbers and types of ports that will reflect in the cost of the chip. Selection of the chip then becomes a choice cost versus performance in this aspect. Next, is the selection of what type of communications will be required for the system. Most chips include an RS-232 port for communications with a PC but there are other ways of communicating with other controllers or devices. These include I^2C, 485, 482, microwire, CAN, and others. The selection of what type to use is based on speed, distance, and protocol to be used. Next, there should be a consideration of what type of language the microcontroller will be programmed in. Typically most microcontrollers can be programmed with some form of assembler language but this is a difficult language to program in. If possible, a cross compiler should be used in a language such as BASIC, Forth, or C. The disadvantage to using a cross compiler is that they do not generate as efficient code as writing in assembler. When memory was expensive and had to interface with this was a serious problem but now that prices and ability to interface with are, cheaper and easier compilers are not at much of a disadvantage.

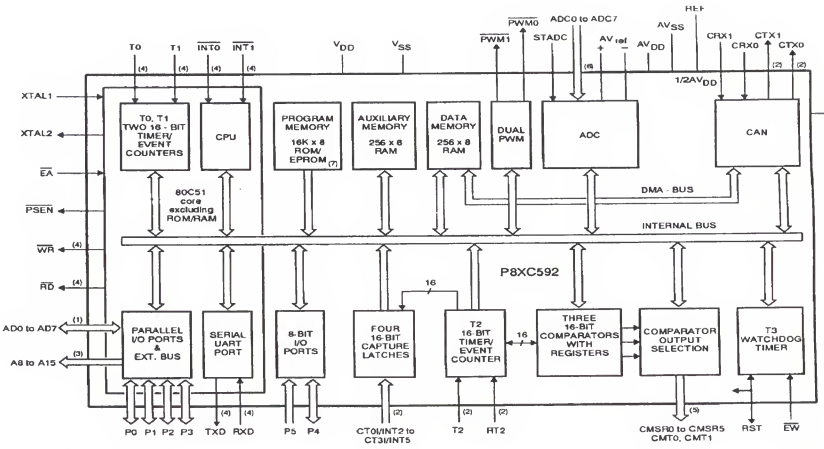
Once all the basic components of a system are known, a schematic can then be drawn of the system. When done with a software package design for schematic layout, the software can check for incomplete connections, improper power connections, and can

convert the schematic to a printed circuit board layout. Specific circuit design of components is not the object of this dissertation and some knowledge of electronics is required to do this correctly. Many circuits can be copied from sources such as electronic manuals and from microcontroller handbooks (Philips Semiconductors, 1995). If a schematic is produced that is not too complicated a beta version can be wire wrapped and tested for functionality. If the design is complex it may be best to have a design house that specializes in such fabrications do the work.

In the system designed for use in aquaculture, all of the board design and layout was done by the author and F. Fialho. The circuits were tested on an individual basis before a complete circuit was drawn up. The basic concept was to design a motherboard that would function as a basis for many different uses in conjunction with daughterboards. The aquaculture system that was to be controlled had several different types of requirements for inputs and outputs. A digital signal was measured that would be pulsed over time, to control the flow meter. This required the use of a counter on the controller. An analog signal produced from a pressure transducer was to be measured. This required an analog to digital conversion. The use of digital temperature sensors required a digital input that needed software control for communications. The control outputs required were of two types. One control function was that of a pulse width modulation output for control of motor speed. The other was an on/off condition for control of certain devices such as feeders, light, or pumps. Another digital output was for the conveying of information about the system. This was done with simple light emitting diodes attached to an output port. Ports were also used for digital input from buttons that served as interfacing devices. One parallel port was used on system reset to read a dipswitch that

was the address of the board. This port was then free to be used by other devices. This was done by the use of a latch chip that was made active on reset and then deactivated so that the signals on that port were read elsewhere in the system. This dual use of a port shows how to multiplex one port for many different uses. Figure 10 shows the basic block diagram of the microcontroller selected for use in this system. The block diagram shows all of the functions that are available to the user on this chip. The understanding of what each of the available functions is used for is essential to the selection of the proper chip. By examining the block diagram it can be determined if all the capabilities are available on the chip selected. In Figure 10 the required elements of the system design are shown as follows:

- 80C51 central processing unit (CPU)
- 2 x 256 bytes on-chip RAM, externally expandable to 64 Kbytes
- Two standard 16-bit timers/counters
- One additional 16-bit timer/counter coupled to four capture and three compare registers
- 10-bit ADC with 8 multiplexed analog inputs
- Two 8-bit resolution Pulsed Width Modulated outputs
- 15 interrupt sources with 2 priority levels (2 to 6 external interrupt sources possible
- Five 8-bit I/O ports, plus one 8-bit input port shared with analog inputs
- Can-controller (CAN = Controller Area Network) with DMA data transfer facility to internal RAM



(1) Alternative function of Port 0.
(2) Alternative function of Port 1.
(3) Alternative function of Port 2.
(4) Alternative function of Port 3.
(5) Alternative function of Port 4.
(6) Alternative function of Port 5.
(7) Not present in P80C592.

Figure 10. Block diagram of 80C592

- 1 Mbit/s CAN-controller with bus failure management facility
- ½ AV_{DD} reference voltage
- Full-duplex UART compatible with standard 80C51
- On-chip Watchdog Timer (WDT)
- 1.2 to 16 MHz clock frequency

(Philips Semiconductors, 1995)

This chip provided all the necessary functions required for the system to be designed. The motherboard concept required that the physical pins be connected to the daugtherboards by way of a header. This reduced the circuitry required for the motherboard and transferred the working I/O to the daugtherboards. The other requirement for the motherboard was that it have on board power circuit so that a regulated source of power could be supplied to the rest of the circuits including the requirements of the daughterboard. The circuit was designed so that any voltage source (AC/DC) from 7 to 24 volts could be supplied to the board would be adequate. A timing circuit was selected to run at 14.7456 MHz. The selection of this frequency was done because it was the highest frequency that could be subdivided down to baud rates compatible with outside equipment. This later proved to be a mistake because although it provided very fast timing for all operations is was mismatched for the internal interrupt timer which clocked the real time system. A lower frequency would have matched all timing requirements. There were jumpers provided in the circuit to allow activation/deactivation of certain control aspects of the microcontroller. The final circuit layout for the motherboard is shown in Appendix A as Motherboard Schematic. The header pinout is shown in Appendix A as Header

Schematic. This schematic shows how to connect the functionality of the motherboard to daughterboards.

The daughterboard design was specific to the required functions of the system. Each daughterboard design had certain goals that were to be achieved. There were two different daughterboards for the system and a third design that was to be used as a large memory module for data logging. The third design was never put into use. The daughterboard called AGUA (Aquaculture General Use Adapter) is used for the general control work in the system. This board was connected to the motherboard by use of the header and ribbon cable. It included six I/O connections by use of RJ-11 telephone type connectors. Four of the connections where through one amp solid-state relays. These relays were used for control either directly of motors or to switch large current solid state relays for high power control. Two of the relays were connected to the PWM output of the microcontroller for control of motor speed in the system. The other two connections where for input of external sensing devices. One the board, two pressure sensors where mounted and routed through proper signal conditioning to the A/D ports for pressure measurement. One other external connector supplied the voltage to the board and the communications. There was a DB-9 connector on the board for serial communications to a serial type device. This was intended to communicate with sensors with that type of capability. The RS-232 connection was used to communicate with a PC to output data and system condition in real time. This proved invaluable in the testing of the system. There were also four buttons on the board. These buttons were used as an input device for changing parameters in the system as it operated. There was also a system-reset button should the system hang up while operating. The board also had output capability for a

Liquid Crystal Display (LCD) but it was not used. The LCD could have displayed local system parameters or acted as a display for interface action. The board also had a real-time clock on board but it was not used. Some important but simple components on the board were Light Emitting Diodes (LEDs) that were used to convey status information. They were connected to the serial port, the CAN port and to power. Looking at the LEDs it could be determined if the power was connected or if information was being transferred. The schematic design is shown in Appendix A as AGUA Daughterboard Schematic. The other design of daughterboard was called CLINK. This daughter board was used as the interface between the system and the PC. This board used the two different communications protocols to transfer information between itself and the PC (RS_232) or information to the system (CAN). This board also held the data from the system in memory until it could be logged by the PC. It had a real time clock with battery backed data storage available but this was not implemented. There was a speaker on the board for creating audible signals for operator attention. A LCD interface was on board but not used. There were two solid state relays for external connection to other devices. This board was connected to the system by a RJ-11 connector. This connector delivered power through the cable to the rest of the system and had the CAN signals on it. The schematic of this board is in Appendix A as CLINK Daughterboard Schematic.

Once the schematics had been designed, it is then necessary to do a component layout and then route the connections between the components. It is important to lay the components out in an orderly fashion so that the least number of routes can be achieved. This task is more an art than science although there is software that can do it. The boards where double sided which means that there are traces on both sides of the board. There

were also components on both sides of the board. This allowed for the placement of external outputs and connectors to be mounted on one-side and circuit components to be mounted on the other. This facilitated a design that was then easy to mount in an enclosure. This was important because of the harsh environment that the system would have to operate.

The board designs were then sent to a fabrication house to be made. Very early attempts were made to do the printed circuit boards in the lab. This proved to be very difficult and not cost effective. Many good fabrication shops can do the work quickly and at reasonable cost. Once the boards were delivered, they required assembly of the parts. The parts were acquired from electronic supply houses. Once assembled the boards were then tested initially without any components to find any shorts and then with each individual component installed to check for proper power connections. If a component gets hot on this first power up the circuit should be checked for shorts or perhaps reverse placement of the component. The boards functioned as designed. They were then installed into enclosures and software testing began.

CHAPTER 8
A MICROCONTROLLED SYSTEM IN AQUACULTURE

A microcontrolled system was set up to demonstrate the capabilities of a microcontrolled system at the University of Florida in the Structures and Environment Lab. The physical system consisted of four 38-liter (10-gallon) tanks placed in close proximity of each other. The basic tank design is shown in Figure 11.

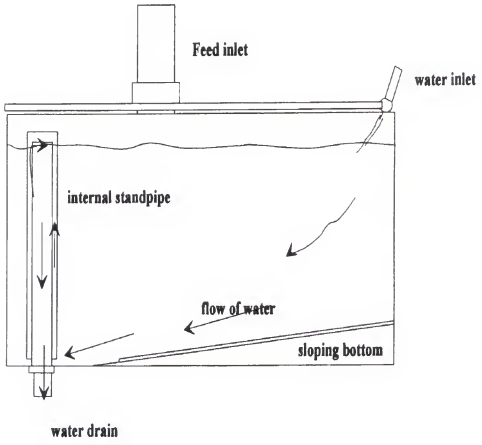


Figure 11. Basic tank design.

The tanks were tied to a common water system with one pump. The tanks were common aquarium tanks except that the bottom was removed and a sloping bottom installed with a standpipe at one end. This standpipe was a double pipe that allowed the water to flow from the bottom of the tank and help remove waste material. The stand pipe design assures that in case of the loss of water inflow there would not be a draining of the tanks.

Pump selection was somewhat difficult as the system had a high amount of solids in it and required a high rate of flow. The first pump used was a Teel flexible vane pump. This pump worked but it was noisy and got hot. Eventually the pump burned up, because it was rated as an intermittent pump and not for continuous duty. The pump motor was replaced with a series wound DC motor that was rated at 10 amps and continuous duty. This motor worked well but the load of the pump was rather high and it ran at almost maximum most of the time. The other main drawback to the pump itself was that it could not obtain a rate of more than 0.7 gpm flow rate. The pump also required that its flexible vanes be replaced periodically. The other main drawback to this pump was that it leaked from its seal. The pump was replaced with a 12-volt DC magnetic drive flooded suction water pump, Hydrothruster model MD412. This pump worked well in the system, had no seals to leak, was quiet in operation, drew no more than 3 amps, and could obtain a flow rate of approximately 1.5 gpm. Neither pump was self-priming which was a problem in the system should flow be lost. With no one to re-prime the pump, the system would not function. The Teel pump would burn up with this condition but this was not the case for the Hydrothruster.

The control of flow water through the system was important in order to maintain the water quality. In order to remove solids from the tanks there was a required minimum

flow rate that was needed. The tanks had sloping bottoms and standpipes with outer shells to facilitate the flow through the tank. This flow would then carry solids out of the tank to a settling filter where some of the solids would settle out. The flow continued from this filter to the main pump that was used to control the flow rate by varying the speed of the pump. The water then flowed from the pump to a flow meter which was the feed back device in the system and then to a biological filter (fluidized bed filter). Flow through the fluidized bed also had a required minimum flow rate as well as a maximum flow rate. The minimum rate was that required to keep the bed fluidized. The maximum rate was set to keep the sand from being blown from the filter. The flow rate of the system was measured by use of a Blue-White digital flow sensor. This paddle wheel type flow meter used a Hall-effect transistor that is activated by small magnets on the paddle wheel. This was important because when the system was running the paddle wheel was jammed by metal particles in the water, which were removed by placing a magnet in the flow of the water elsewhere in the system. The flow meter was very accurate but had a tendency to get stuck due to the high solids content of the water. It is also important to be sure, the orientation of the meter was correct. In this case, the meter needed to be placed vertically for proper operation. The meter was the smallest available and in larger sizes may not have had the same problems.

The system had a settling filter and a fluidized bed biological filter. The settling filter was a multi-purpose device. It was used to equalize the system hydraulics. Having the tanks individually drain into a common open collector does this. The collector was a 6-inch pipe closed at one end and standing on end as shown in Figure 12. The bottom was

shaped like a cone to concentrate the effluent as it came through the filter. On the bottom was an outlet that is piped to a pump that was used to remove solids accumulation. This

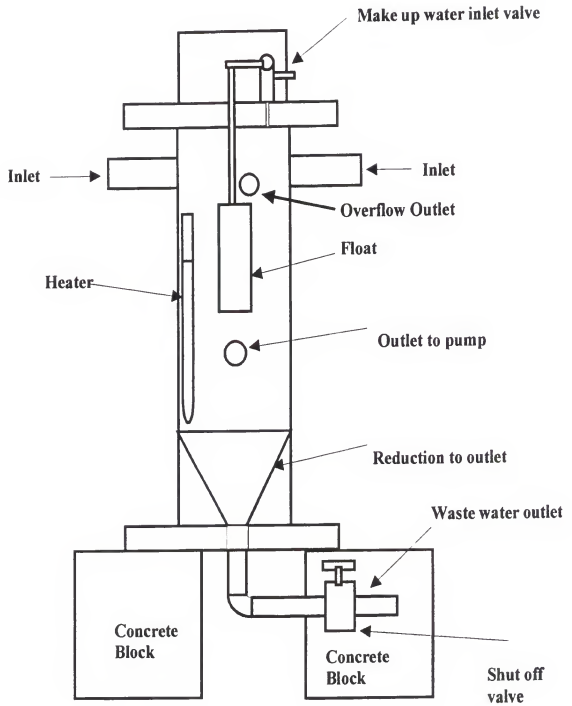


Figure 12. Settling filter.

outlet had a shut off valve to facilitate cleaning of the pumps or isolating the filter. A heater was placed in this unit since it was common to all tanks. A temperature sensor was also placed here to measure the water temperature. A make-up valve was attached to a float that would open the valve when the water in the system was removed. Located near the top was an outlet to allow water to flow out when the system pump was shut down. The tanks would accumulate an excess of water due to the pumping in the system and when the pump was turned off this excess would return to the filter and flow out of the system. This occurred each time the system waste pump was activated. The filter was set on concrete blocks to elevate it off the floor and allow for clearance for the waste line.

The bio-filter was an Aqua-Tropics Quick Sand fluidized bed filter. It was located after the pump and flow meter. The filter contained approximately 0.25 kilograms of 20/40 sand mix. It proved adequate in early tests but when the system was loaded to capacity, it failed to function as designed.

A Motorola differential pressure gage MPX 5100DP is used to measure the pressure drop across the bio-filter. This is an on-chip signal condition, 0.2 v to 4.7-v output, temperature compensated, and calibrated fully integrated pressure sensor. The on-chip capability is that there is a four-stage amplification network, incorporating linear bipolar processing. This gives a compensated output signal that is directly readable by the microcontroller. Since this is a dual port unit it is used with a port connected to each side of the fluidized bed filter, thus showing the pressure drop across the filter (Motorola, 1996).

There is a second pump for removing wasted water from the system. This is a 115-v AC centrifugal pump that is connected to the bottom of the settling filter. Its job is

to pump out waste that had accumulated at the bottom of the filter. It pumped the waste out of the system to a drain. The pump is activated by a solid state relay that is connected to the microcontroller. The microcontroller is given commands by the network via CLINK and from the computer. As water was removed from the system, a make-up valve was activated by a float. This then provided an exchange of fresh clean water into the system in order to try to maintain water quality. This was important in water quality management, as the system had no means of removing the build up of nitrates in the system. The control of the percentage to be wasted each day was set by the operator based on his chemical analysis of the water. Periodic measurements where made in the levels of NH_3, NO_3, and NO_2. Based on these measurements, the percentage of exchange was either increased or decreased. The PC calculated the timing for the waste pump by knowing the volume of the system, the percentage of exchange, and the flow rate of the pump. The PC also calculated the time intervals required between each of the exchanges. It would compare these times against the system clock and at the appropriate time send the command to turn the pump on and give the amount of time for the pump to run.

There is a DC20 oil-less diaphragm air pump for aeration of the system. This pump is controlled by a solid state relay that is in turn controlled by the PWM on a microcontroller. The pump rate is based on the measurement of dissolved oxygen in the system. A Royce 9000 model DO meter measures the dissolved oxygen content of the water. This meter is interfaced to a microcontroller by connecting its 0-1 v output to an A/D channel. A Motorola MPX5100 pressure sensor measures the air pressure. Overhead florescent lights provide the lighting. One 200-watt aquarium heater heats the water. Dallas Semiconductor 1820 digital temperature sensors are used to measure the

temperature of the water and air. The basic layout of the systems components is shown in Figure 13.

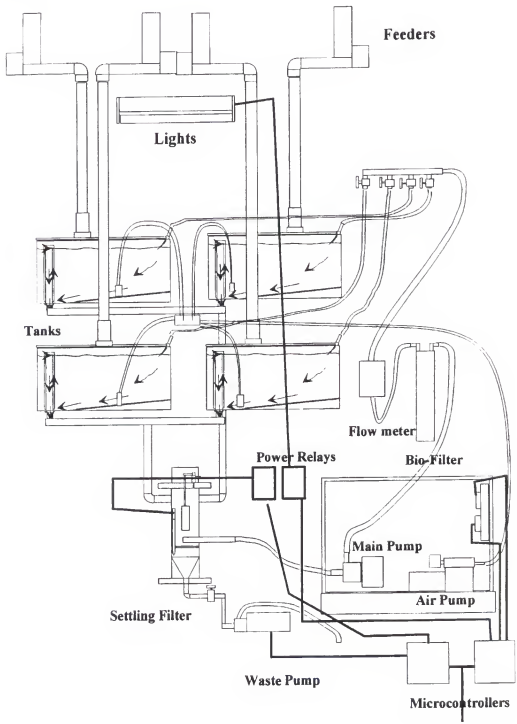


Figure 13. General overview of system.

A microcontrolled system designed in the lab is used to monitor and control the system. It is based on Philips 80C592 8-bit microcontroller with on-chip CAN. The system uses two separate controllers that were linked together by a CAN protocol which is then linked to a PC with another microcontroller system by way of RS-232 serial port. The PC monitors the system with a front end developed in C++ Builder (Borland Inc. 1997).

The microcontroller system was developed with a single design for a motherboard that had the necessary I/O for interfacing with a specifically design daughter board. The motherboard used the 80C592 chips with a 64K EPROM to hold the program and a 32K DRAM chip to hold the working memory. The chip has 8 analog to digital channels with 10-bit resolution. The clock crystal was a 14.7456 MHz that meant that the system could run at over 1 MIPS. The board had a voltage regulation circuit which allowed input voltage to vary from 7 to 24 V DC/AC, which meant there were any number of ways to power the unit. The voltage is regulated and filtered for all additional circuitry. The motherboard also has a header that allowed the setting of an 8-bit address by the addition of jumpers. A Philips CAN physical layer interface chip is on the board to allow network communications between all other boards. All ports necessary for I/O are connected to a 40-pin header that allowed the interfacing to a daughterboard. The 80C592 have three 16-bit timer/event counters. Timer 0 is used for timing of the RS-232 communications at 9600 baud, Timer 1 is used as a counter for the flow meter, and Timer 2 is used as a system timer which gave a resolution of 0.05 seconds. The timers are independent of the program logic that is to say they are interrupt driven which means they are free running and not affected by delays from code implementation. This is important so that timing schemes for the operation and controls of the system are unaffected.

Two AGUA daughterboards and a CLINK daughterboard were used in the system. One daughterboard (CLINK) was designed for interfacing the system to a PC This motherboard with the address of 0 was a CLINK. The requirements of this board are to receive communications from the PC and to relay commands to the system controllers. This is done with a serial communications link to the PC at 9600 baud. This allowed for ASCII type protocol which made messaging very easy to understand. All messages are sent in a word type command with data sent in hexadecimal form. It also receives data from the system controllers, which is then relayed to the PC. It acted to relay messages from one control unit to another when conditions required. This is done with the on-chip CAN that is a multi-drop type network. The arbitration of who is sending and who is receiving is done by hardware with its own error detection. This simplified the transmission of messages. Messages are broadcast on the network with the appropriate address so the receiving device would know to react to the message. When the address of the message was 0, it is a message back to CLINK. It also had two external digital outputs, one of which is connected to the heater system for the water and the other is connected to an alarm. The other daughterboard (AGUA) is used for system control. It consisted of four digital outputs, two of which are also PWM. It has one digital input for a temperature sensor and another input which doubled as a digital and an analog input. Each board has three LED light indicators, one for power, one for communications traffic, and one for error conditions.

The two AGUA daughterboards used are configured slightly differently. One acts as a controller for the system, AUGA-controller (AGUA-con) and is used for feeding the culture, AUGA-feeder (AGUA-feed). The motherboards with the addresses from 1-127

are to be AGUA-feed and the motherboards with addresses from 128-255 are to be AGUA-con. This gives the ability to address up to 255 AGUAs. AGUA-con received input from the flow meter in the form of pulses. These pulses are counted, compared to a clock, and used to calculate the flow rate of the system. Flow rate is then subtracted from the required set point. The difference is then compared to a control band and corrective measures are taken if the value is outside the band. Taking the value, multiplying by a proportional constant, and then adding to the PWM value does this. If the value is positive it increases the motor speed, if it is negative, it decreases the speed. Due to the fast nature of microcontrollers, a time constant is used to delay the fluctuations of the pump and to allow a steady change in the flow rate. The air pump is dealt with in a similar manner, except that the set point is derived from an analog signal from the DO meter that is converted to a digital one in the AGUA-feed unit. This value is sent to CLINK that then sent it back to AGUA-con. AGUA-con would then make the necessary calculations and vary the PWM of the air pump. There is logic in place to keep the pump from ever shutting down completely and the time constant is very large for this procedure as it often took a long time to bring the O2 level back up to the required level. AGUA-con is also used for turning on/off the lights (photoperiod) for the system. This is done with simple digital control although the PC does the logic. The PC has an algorithm for calculating the sunrise and sunset for any day of the year at any latitude and longitude. The coordinates and day of the year are entered into the program (the PC would use its own time) and the PC sends CLINK the signal for switching the lights. CLINK then sends the message to AGUA-con, which then controls the lights. AGUA-con and the PC control the waste pump. The PC is used to enter the percent water that will be removed daily, the interval

between dumping, and water quality requirements. A regular scheme of wasting water is set up to allow for the discharge of nitrates from the system. Water quality parameters are measured externally and input into the PC. Based on these measurements, the water exchange needs are determined. The water is made up by a mechanical float system in the settling filter. AGUA-con also measured the air temperature and sent its value back to the PC via CLINK. AGUA-con also had two pressure sensors to measure air pressure and the pressure differential across the fluidized bed filter. The air pressure sensor is used to detect conditions such as loss of air pressure or low air pressure. The differential pressure is used to observe fluidized bed behavior and acted as a back up when the flow meter was stuck. It works when the flow meter reading is zero and the PWM was positive. When this condition occurs the flow meter is assumed to be stuck and the differential pressure is maintained at its last average reading, using the same control scheme as the control by the flow meter.

AGUA-feed has different logic in order to carry out the feeding scheme of the culture. The basic concept is for the PC to have a fixed feeding schedule inputted and then based on the time from its clock it would send the feeding signal to CLINK that then relays the messages to AGUA-feed. The message includes the feeder that is to be used and the amount of time that the feed is to be delivered. The feeders are calibrated based on the amount of feed delivered per time. The accuracy for feed is based on 1/4-second intervals, which is the smallest amount of time that a feeder could be turned on. The amount of feed delivered is based on the percent of body weight of the culture. A growth curve is used in the PC to estimate the mass of the culture and the appropriate timing is calculated for each feeding. This growth curve is generated using data and parameter from FishBase 97

(Froese and Pauly, 1997). Although there were two pressure sensor inputs on AGUA-feed, one was not used, and the other is wired to convert the analog signal from the output of the DO meter. This signal is converted to a digital signal and then sent to CLINK where it is relayed to AGUA-con who had control of the air pump. The temperature port on AGUA-feed is used to measure the water temperature and relay the information to CLINK that had control of the water heater. Since the system is set up in an air-conditioned lab, heating is always necessary for the culture.

Software Development

The development of a computer control system is dependent on the code generated for that purpose. The code must be error free and logic sound. In this system, two different compilers were used to generate the code required. C and C++ was the language used for the system.

One of the major difficulties in the development of microcontrollers has been the writing of the program code for the controller. This was due to the use of assembler language in development of microcontroller chips. Until recently there was no other choice for code development. In the last few years C code cross compilers have become reliable and generate small enough files to be used. C code was used for this microcontroller development. A Boston Systems Office Tasking (BSO/Tasking 1995) compiler was used to convert the code from C to assembler and then link to Intel hex code which was downloaded to the memory onboard the microcontroller motherboard. The motherboard system used a 64K EPROM memory chip for program memory. This required the burning of the EPROM each time the code was changed. This was a tedious process and requires the removal and installation of the EPROM. The use of an emulator

would have expedited the process. The large size of the memory chip allowed all of the code for all devices to be put into one source code. The code that was used was determined by the address given to the device. This simplified the code development, as much of the code was common between devices. The first significant block of code had to do with setting up a system timer. This timer is what was used for the counting of real time that caused events to occur. The other blocks of code had to do with communications, both the serial port and the CAN port, and with control of the system process. This code was not a large file and when compiled and linked became less than 22K in a hex file. This means that there was more memory available to extend the logic and control functions. A flowchart of the code can be viewed in Appendix B.

The computer interface was developed using Borland C++Builder (Borland Inc., 1997). This is Windows graphical user interface (GUI) application for use with Microsoft Windows 95 operating system. C++Builder applications are form based. Every window and every dialog is a form. This means that the application can be made visually with only the working code having to be written. The basis of this compiler is the C++ language. The windowed environment allows different screens to be used for specific goals in the system. The software is tied to the microcontrolled system by way of the serial port. The use of timers in the system, carried out the routines required for updating data and sending control commands in a sequential and systematic method. This code was developed and tested in the lab with the microcontrollers. There are over 500,000 lines of code in this interface. Appendix D has the source code for the interface.

Interfacing the System

One of the most important aspects of a system is the interfacing of the system to the real world. This includes many different level of interfacing; the following are the levels described in more detail from the top down.

- Human to PC- this is perhaps one of the most important interfaces as the system needs to relate information to the operator who is the final controller. The typical computer has a monitor for output and a keyboard and mouse for input. This is adequate for most applications. There can be additional output devices such as sound or a printer. The PC interface was developed using Borland C++Builder software. This is a visual development package that produces a windowed environment. The human-computer interface is a critical factor in developing working models. If the model is too complex to be understood and the interface is not self-explanatory, the system will not be used. The interface used on this system is divided into various windows that dealt with only certain aspects of the control procedure. The sub-models were developed within the system to solve the specific task of each control parameter. The system primary screen, shown in Figure 15, is used to show the real-time values of the system as it operated. It allowed user input to change the set point values and carried out the necessary procedures to carry out these changes. There are additional screens to give graphical views of processes as they occurred. The predictive screen for growth that drives the whole system is both numerical and graphical. The screens are color coded so that associations between systems could be made at a quick glance. The PC also acted as a

repository for storing data as it came in from the system. This data could then be analyzed for fine-tuning of the system by noting trends and erratic behavior. The system would function without the PC being connected, although not to its full capacity.

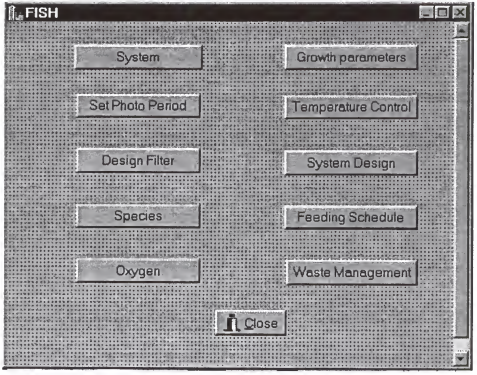


Figure 14. Opening screen.

The use of a graphical user interface (GUI) is important as people relate to this type of interface much better than a text driven scheme. The layout of this interface should be done by levels of importance. The first screen should always lead to the other sections and be as simple as possible. Figure 14 shows the opening screen to the system described. This screen appears upon the clicking of the program icon. It stays on the screen regardless of what other screens are brought up. It can then interconnect to other parts of the program without having

to find the path there. The clicking on one of the buttons then brings up some other functionality of the system. There is an initialization of the system that must be carried out in order to establish all of the parameters for the system. The buttons: Set PhotoPeriod, Design Filter, Species, Oxygen, Growth parameters, Temperature Control, System Design, Feeding Schedule, and Waste Management are used to initialize the system. Each of these functions is explained in the following chapter. The next of importance is the system status screen as seen in Figure 15.

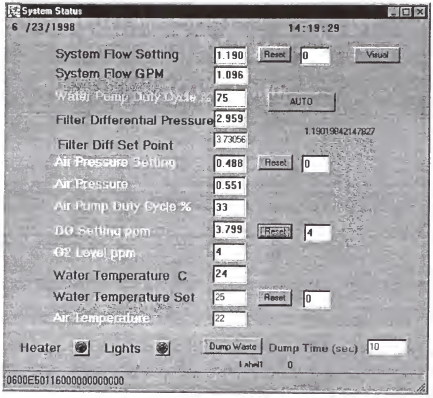


Figure 15. System status screen.

This screen gives the important system parameter readings so that the system can be checked for operating status. The primary screen should be observed by the

operator at most times. It also allows the operator to change the parameter set points at his discretion. It has simple status indicators that are lights on the screen to indicate the status of the heater and lights. A simple glance can indicate the status of these devices and reduce the observation time of the operator. This screen is over laid on the previous screen and access to that screen is accomplished by the click of the mouse. The operator can then access other parts of the program. There is a need not to put too much information or interaction on each screen. When possible it is best to make the input by use of a mouse and simple decisions such as on/off, Dump Waste, or Manual/Auto. The need for keyboard input quite often invites errors to be input to the system, although this is sometimes unavoidable.

- PC to system-The next level of interfacing done on the system is from the PC to CLINK. This involves the use of a serial port on the PC that connects to a serial port on CLINK. This is the main link between the system and the computer. Information from the system is obtained through a request from the PC. Periodically the PC generates a signal that is a request for the data from CLINK. This system is automated as a timing sequence in the PC code. The PC can receive intermittent information from the user. This information, in the form of parameter settings or control commands, is then sent to CLINK who processes the information and the relays it on to the proper system microcontroller (AGUA). Certain timed sequences occur in the PCs operation that generates control commands that need to be carried out by various control systems. These commands are involved with the photoperiod, feeding, and waste management.

The system relies on the computers time base to generate these commands. Should the computer fail to operate these functions will not occur.

- System to system-the next level of interfacing is done from CLINK to the AGUAs. CLINK is responsible for transmitting and receiving data and commands from the AGUAs. This communication between microcontrollers is done via the CAN interface. CLINK uses a timing sequence to periodically ask for data from the AGUAs. The data is stored in memory and then relayed to the PC on request from the PC. CLINK is also responsible for routing control commands from the user or PC to the proper AGUA. When a command is given CLINK arbitrates who the command will be sent to by using either the address given or internal logic to decide the AGUA responsible for the action. CLINK also receives information from the AGUAs that is acted upon in several ways. If the information received is an error message from an AGUA, it is relayed to the PC for operator action. If the information is data, which is needed by another AGUA it is the routed back to the proper AGUA. CLINK also was used as a controller of the heating system for the water. In this case CLINK would receive temperature data from AGUA-feed and use this data to decide the status of the heater, either on or off.
- System to sensors and control-the next level of interfacing was done by the AGUAs. This was the communication of sensor data and control functions and interfacing to CLINK via the CAN. The AGUAs were the central part of the system. Once they were set up, they would continue to function without information from the rest of the system. The system would not function as design if this should happen but each AGUAs control function would continue. The

AGUAs used both digital and analog inputs to gather data and control processes. The internal clock on each AGUA was responsible for timing operations that occurred. Each AGUA had sensors connected that it was to interface with and record the data taken. This data was stored and then relayed back to CLINK on request. The outputs from each AGUA for system control were solid state relays which either directly activated the process or activated a larger relay in order to handle the higher current required.

- Sensors to human- this level of interfacing is a very basic but required level as it is from mechanical sensors that give a direct readout to the operator. This includes such devices as thermometers, air and water pressure gauges, and level indicators. These types of devices are very basic and are used to compare the readings that the electronic system may be reporting. They require a physical presence of the operator to manually take the readings but they are generally very reliable readings.
- System to the world- this is a very new concept in computer monitoring and control. The PC was put on an active link to the Internet. Using software by Netopia Inc. called Virtual Office (Netopia Inc., 1997) the system was able to become an interactive link on the Internet. It was then possible to link to the system PC from anywhere that the Internet access was available. This allowed the user to not only monitor what the status of the system was but also to control any function that was available through the PC interface.

Table 8. System control of functions.

Controlled Function	Sensor	Proportional	Logical	Setpoint	Timed
Flow of water	Flow meter	X			
Flow of air	DO meter	X			
Flow of waste			X		X
Flow of feed			X		X
Photoperiod			X		X
Temperature	Digital			X	

The linking of the control of the system to the way in which it was controlled is shown in Table 8. The flow control of water and air was done through proportional control that was achieved in a software algorithm. The control of photoperiod, waste removal, and feeding was a timed event that then was an on/off event for the microcontroller. The temperature control was done through a simple setpoint check periodically.

CHAPTER 9
RESULTS AND DISSCUSSION

Computer control of a biological system is not an exact science. This series of experiments to develop a complete control system demonstrated that it could be done. The process was not a simple set of steps but an involved process of learning and adapting to the system requirements. The ability to establish a system of biological control is certainly within the scope of this equipment.

Growth Prediction

The fish that were grown in the system had a growth pattern that reflected what was found in literature. One of the primary functions of the system was to maintain the water quality in the system. The main cause of poor water quality is the build up of waste products from the fish. The best way to control the build up of waste is through proper feeding. Unconsumed feed is the main reason for waste products to occur. If the feed input is too high, some of the feed is not eaten and it decomposes to waste. If the feed amount is too low, there is poor growth. The amount of feed delivered is determined by the weight of the fish. The process of feeding the fish should then be based on the total biomass in the tank. As the fish grow, their total biomass increases which leads to the development of a growth curve for the prediction of growth. In this system, the von Bertanlanffy growth model was used. This model was adjusted to match the growth rate

found in the study. This was done by matching graphs with the use of MathCad software (MathSoft Inc., 1995). The results of the match are shown in Figure 16. This shows all of the tanks used in the study. Each tank had a different age of fish so growth could be compared over a broader spectrum of time.

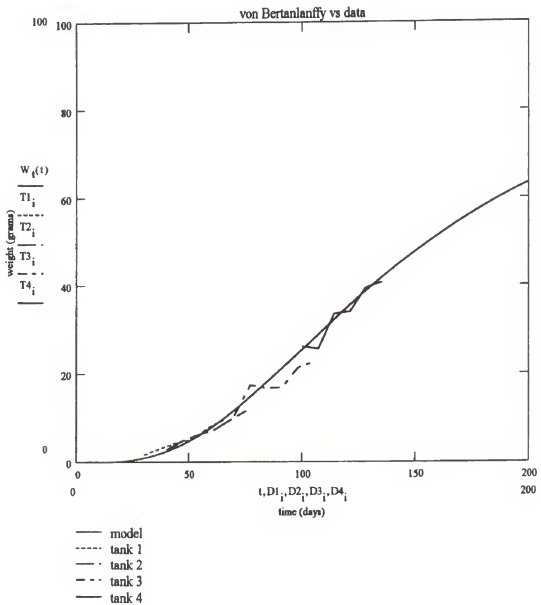


Figure 16. Model vs Data.

The resulting analysis gave growth parameter values K=5 and b=4.5 from the model, the graphical comparison is shown in Figure 17 for a single tank and the model. This early growth had a much better fit than later growth using these parameters. The correlation of the tanks to the model were for each tank: Tank1= 0.993, Tank2= 0.994, Tank3=0.943, and Tank4=0.951.

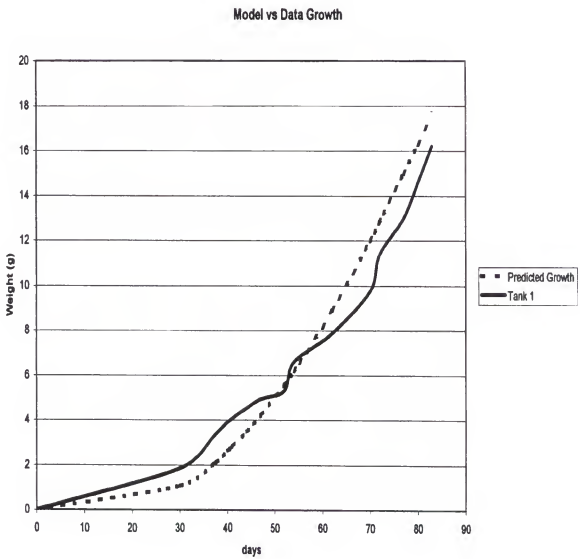


Figure 17 Model vs tank 1 data.

With the software interface, the same results can be observed by using the Growth button to run the same analysis. The ability to change the parameters for the fish and match to known data lets the user create growth curves for fish of which there is no known growth data. The software also adjusts the system to feed in the proper manner for the optimum growth and to maintain good water quality. The screen that shows the data input for the growth function is shown in Figure 18. The growth of fish predicted by this curve does not fit the entire life cycle. Since there are different phases of growth, it is necessary to correct the growth period for each different cycle for that species.

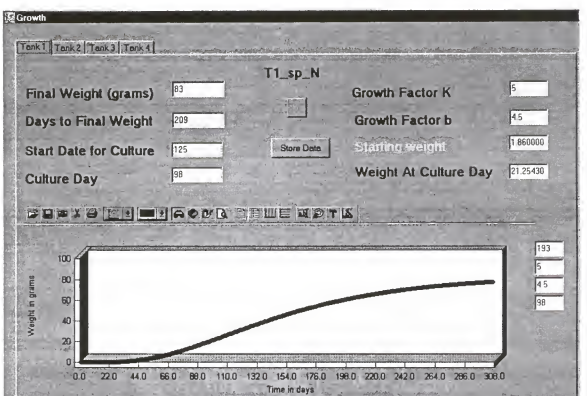


Figure 18. Growth screen with early growth prediction.

From the screen viewed in Figure 18. It can be seen that the maximum growth expected for this curve is 83 grams. This factor worked well for the early growth of tilapia but the

final weight of tilapia can exceed 2 kilograms and the curve should be adjusted for that stage of life. Better data needs to be obtained and studies done to find what the growth parameters are for these stages. In Figure 19, the growth has been adjusted for the final grow-out of the fish. This data is used by the program to calculate the amount of feed going into this tank. This dynamic change is one of the more important aspects of the system control.

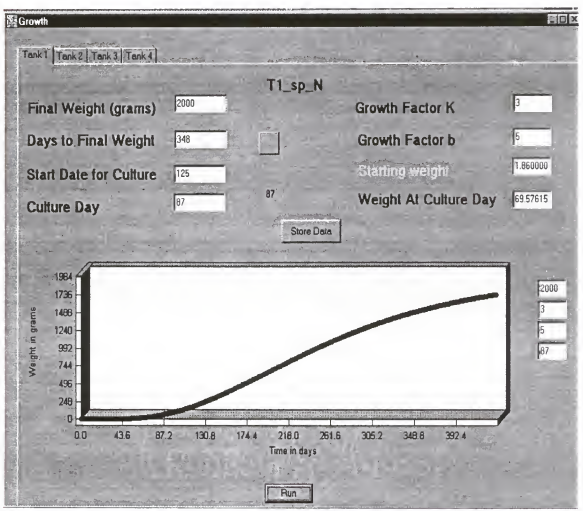


Figure 19. Growth screen with later growth prediction.

From Figure 18, it can be seen that when the growth is calculated differently with new parameters, a much higher growth rate is predicted. The final weight is now at the level desired for the full-grown fish. The changing of parameters could be done automatically with a more complex algorithm but this would also require more knowledge and data to come up with the proper scheme. The ability to predict growth and then have it interact with the system was done at a reasonable level. Longer-term studies should be done so that the data could be utilized in a system such as this.

Control of Flow of Water

Control of the flow rate was done by first selecting the rate from the system screen as shown in Figure 20.

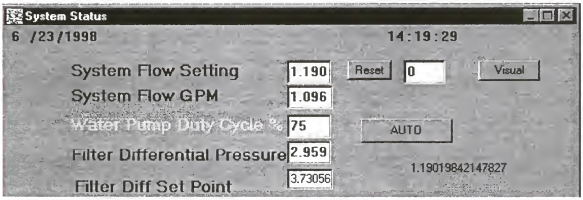


Figure 20. Flow control section of system status screen.

Flow through the system was logged and the data were then analyzed to see what scheme worked the best. Several different trials were made. When the flow meter was working properly, the automatic mode worked well. There were problems with keeping the flow meter from being stuck. The cause of its ceasing to operate was generally sand from the filter and then sometimes a buildup of waste from the system would stop it. Better filtration would probably have solved this problem or perhaps another kind of flow meter.

The flow through the system was maintained through a series of backup systems so that if one failed, such as the flow meter being stuck, the secondary system would take over. Other than when a pump failed, the flow system function as planned. The main use of this flow system would be in research in filter design and system flow design. The purpose of this work was not to study filter design but it would certainly be a useful tool in that study. Some of the data is shown in Figure 21.

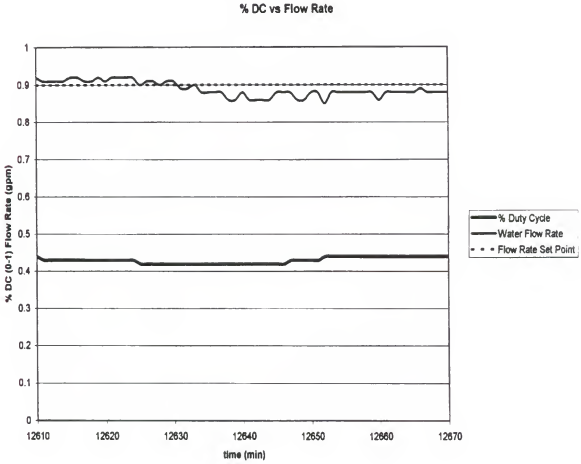


Figure 21. Percent Duty Cycle versus water flow rate.

The time span in Figure 21 was one hour during which the flow rate was set to 0.9 gpm. The percent duty cycle varied from 42 percent to 45 percent while the actual flow rate was

0.86 gpm to 0.92 gpm. The capability of the microcontroller to maintain the flow rate within 5 percent of the set point was well within the design of the system control. In Figure 22, the duty cycle has been set to manual control to observe the fluctuations in flow rate. The duty cycle is fixed at 50 percent but the flow rate varies from 1.2 gpm to 1.35 gpm, approximately an 11 percent fluctuation. The periods when the flow rate goes to zero are due to the waste removal cycle when the main pump is shut down. If at any time the operator wanted to view the real time response of the system the visual output view could be brought up

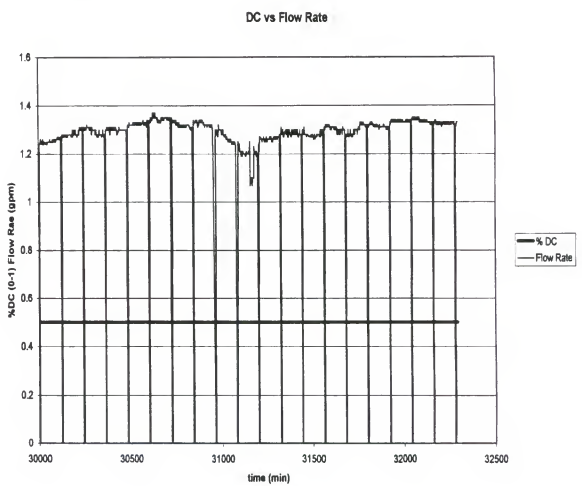


Figure 22. Fixed duty cycle versus flow rate

and a real time graphical output showing the parameters could be viewed. This is shown in Figure 23. Figure 23 and shows not only the flow rate of the water but also some of the other critical parameters involved with the system such as dissolved oxygen, filter pressure, and system temperature. This gives a quick view of how the system is functioning and allows the operator to view any changing patterns in the system.

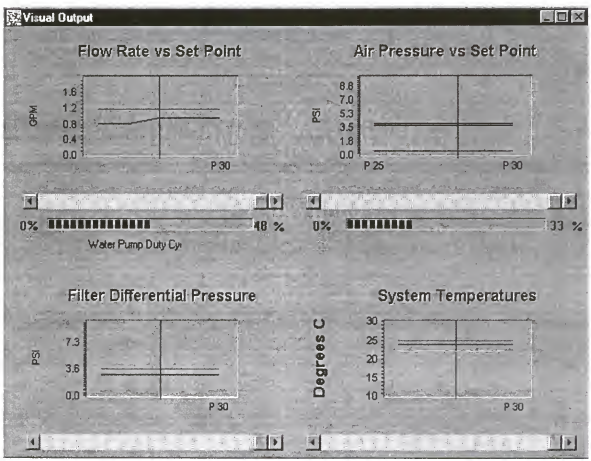


Figure 23. Real time graphical output.

Control of Oxygen

The control of the level of DO in the water was accomplished by a similar method for flow control. It was a bit more difficult due to the large time lag that was involved in increasing or decreasing the level. Using the software, the operator could find what the maximum level of DO could be expected from indigenous conditions. Figure 24 shows the

calculation screen that was available to the operator. When the operator input the ambient conditions and found the maximum level one could then enter the parameter settings in the system screen. The PC would then relay the information to the micro controllers and they would take over the control of the air pump with feedback from the DO meter. The large time delay caused the pump to gradually come up or down to desired speed.

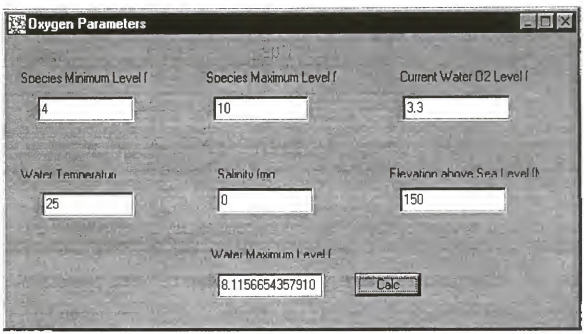


Figure 24. Dissolved oxygen calculator.

The logic in the controller functioned as designed and included backup methods to deal with equipment failure. The DO meter used did not function very well as it was removed from a previous project and in somewhat questionable condition. The probe was subject to fouling that meant that it had to be cleaned very often and when this was done, the readings would then change to a higher reading. The readings from the meter were not accurate as comparisons to known DO readings were made. This meter did function in relation to the DO level in the water but not to acceptable standards. It was used so that a relative value could be used for control, which it did provide. A better system of reading

DO should be found perhaps by automating the DO probe to keep it out of the water until in use, a very good application of microcontroller technology. By using data as depicted in Figure 25, the time lag can be observed between the time that the set point is changed and the reading is corrected to that value.

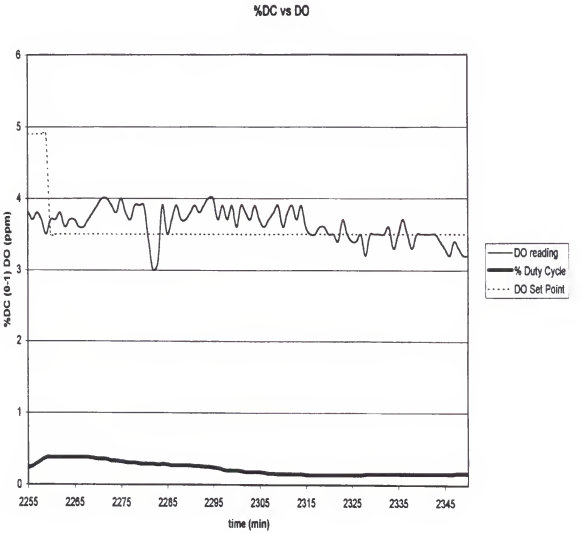


Figure 25. DO readings versus the percent duty cycle.

The pump speed was adjusted to match the required relative level. When run for long periods of time the fouling of the probe would make the DO useless as input. After an initial period, its values were no longer used and a safety value for pump speed was used.

Control of Temperature

Temperature control was done with a 200w heater installed in the settling filter. There was a digital temperature sender in the inlet stream of the settling filter. This sender would measure the temperature and send data to AGUA-feed. The data was then transferred to CLINK where logic was in place to compare the temperature against a set point and then turn the heater on or off accordingly. This method used a simple deadband of 1 degree. If the temperature fell outside the dead band then the heater was turned on or off. The heater was controlled by a separate power module, as the current was greater than the capacity of CLINK. Some of the data from the system is shown in Figure 26.

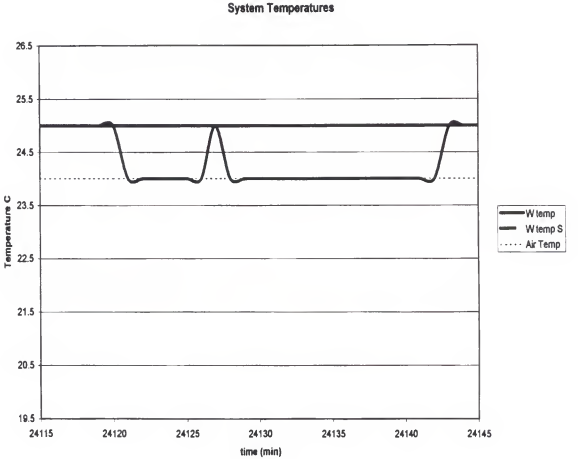


Figure 26. System temperature.

A second set of data is shown in Figure 27. This data is a longer run of data from the temperature record. The value of this section of data is to note the spikes that randomly occur. These spikes are out of the range of the sensing device and can be attributed to either hardware or software problems. The most likely cause was a communications problem that did not show up in casual observance. This type of random problem is very difficult to solve. The non-severity of the problem allowed for the process to continue to run without any damage. In a more severe case, some action might have been required.

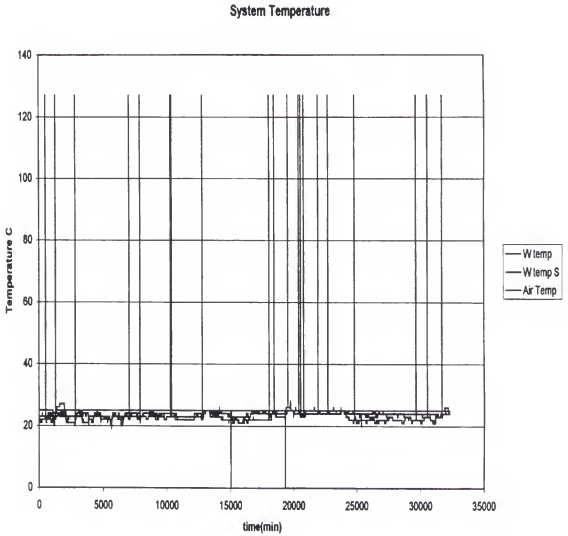


Figure 27. Noise spikes in communication.

Photoperiod Control

The photoperiod was controlled by the PC using its real-time clock to keep track of the time and the model to determine when it was day or night. The system used information supplied by the operator as shown in Figure 28. The position on the earth in which to simulate the photoperiod was input and the date set on the PC. The sunrise, sunset, and length of day would then be computed.

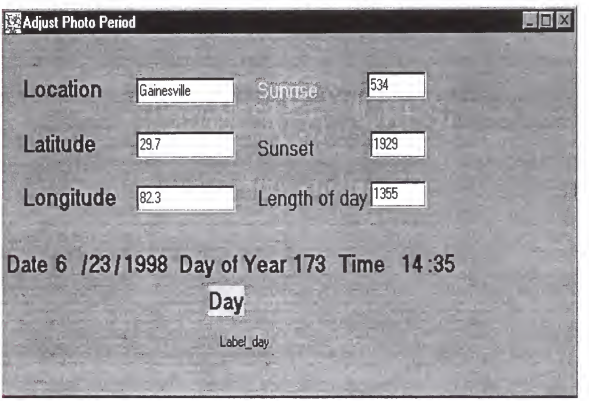


Figure 28. Input screen for photoperiod.

When this information was determined, the PC would send a message to CLINK who would in turn route the message to AGUA-con. AGUA-con would then turn the lights on or off as required. In Figure 29 the period of day and night can be seen with the 1 value being day and the 0 value night.

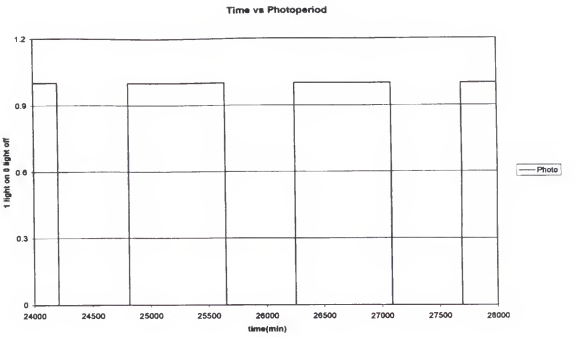


Figure 29. Photoperiod data.

Filter Design

Filter design is a major concern of the aquaculture research. Although the design of a filter was not the purpose of this study, data was taken to show the effects of flow rate on a fluidized bed type filter. This data would be useful in the study of filters and their design. In Figure 29, the differential pressure of the filter is shown over time. The duty cycle of the pump was held constant during this test to show the fluctuations in the filter.

Reliability

This study was done to show how with a methodological approach a complex computer system could be designed, built, and operated. The system proved to be reliable when most of the hardware and software bugs were removed. Although it functioned well, there was much room for improvement. Many problems were unforeseen until the system was assembled and only some of the major problems were corrected. The fish that lived in the system thrived and grew despite some harsh conditions they were subjected to when

equipment failed. There were few equipment failures in the electronics of the microcontrollers. They had a very robust design and survived many accidental spills and

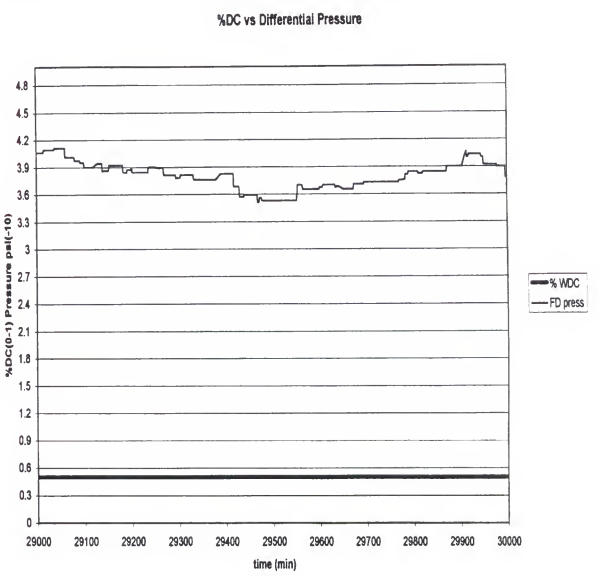


Figure 30. Differential pressure readings.

drops. This reliability can be attributed to the electronics industry. The quality control in most component manufacturing plants is exceptionally high and when assembled into a system carry over that reliability. The ability to network the microcontrollers into an integrated system improved the system reliability.

CHAPTER 10
CONCLUSION

A seven-step methodology was created that gives a general procedure to develop complex computers systems in agriculture. This methodology can be used as a procedure to systematically design other agricultural control applications such as irrigation systems, greenhouse environment, or animal environmental enclosures.

The methodology developed had the following seven steps to build a system:

1. Describe the overall system objective.
2. Break the system into independent functional subsystems.
3. Define and describe each subsystems process.
4. Develop or use an existing model.
5. Define the interactions between subsystems.
6. Categorize the exchange of materials, energy, and information.
7. Integrate all of the subsystems into one system.

Methodologies that are presently in use today involve very general solutions for broad problems. These methodologies can be very useful when trying to solve problems such as population dynamics or designing computer programs. The methodology developed in this study solves problems that deal with both agricultural dynamics and computer control strategies. It integrates both software and hardware into a functional system. The methodology's usefulness is demonstrated by the creation of an aquacultural system.

A microcontrolled closed aquaculture system was developed using the methodology described in this work. The system was designed, built, and tested in a lab at the University of Florida. The system was run for approximately 4 months and showed excellent results with respect to its ability to adjust to the dynamic environmental changes that were a characteristic of the system. Building systems that are complex and interactive is made easier using distributive networks. A comprehensive methodology should be followed when trying to develop non-stationary dynamic processes to reduce the inherent complexity that will develop from interactions that occur in these systems.

Future concerns that need research include networked systems, intelligent controls. These are areas in need of further development, that of a reliable oxygen monitoring, feedback and control system. (Losordo et al., 1989). Design of equipment for use in harsh environmental conditions, power monitoring and control, temperature monitoring and control, micro management of support equipment, photo period control, and continuous monitoring systems.

The adaptability of these types of systems makes them very attractive for future development. There are many different areas of agriculture that could benefit from the development of computer controlled systems. The fast changing pace of development in the microcontroller field means that designs need to be put together in a reasonable time so that they will not be obsolete. The system that was developed in this study is already out of date and newer and better chips are available for development. The fact that these system are generally under a software type control means that they can be updated quickly as better control schemes become available or errors are found that need to be corrected. The other advantage to the development of these systems is their cost. The price of chips has continued to fall even as they are improved in speed and performance. The field is rapid developing and new standards for

these systems need to be developed such as the implementation of ISO 11783 for distributed control systems.

Methodologies to develop complex systems are only a guideline to the successful design of such systems. The final system can only be validated by field testing and debugging. This methodology is designed to assist in the development of agricultural systems when integrated with computer control. The methodology is a way to stay even with the rapid development of technology and its implementation. This study's objective was to create such a methodology and demonstrate how it could be used. The methodology was developed and the system developed using the methodology worked as design. The methodology can be used as a tool to design control systems for other complex systems in agriculture and aquaculture.

APPENDIX A
SCHEMATICS AND LAYOUT

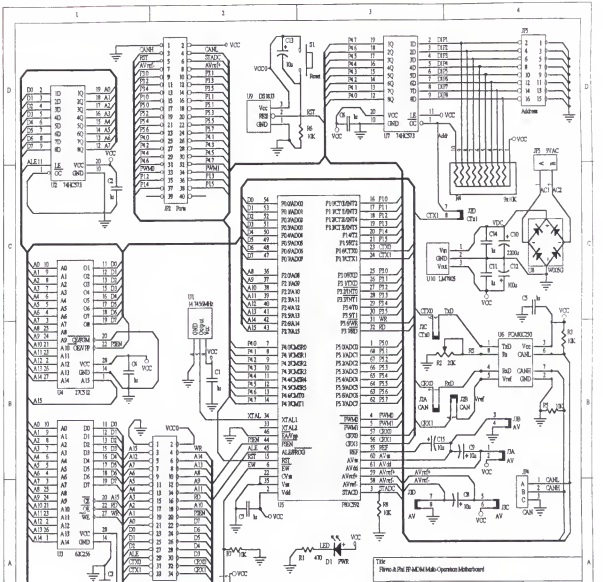


Figure 31. Motherboard schematic

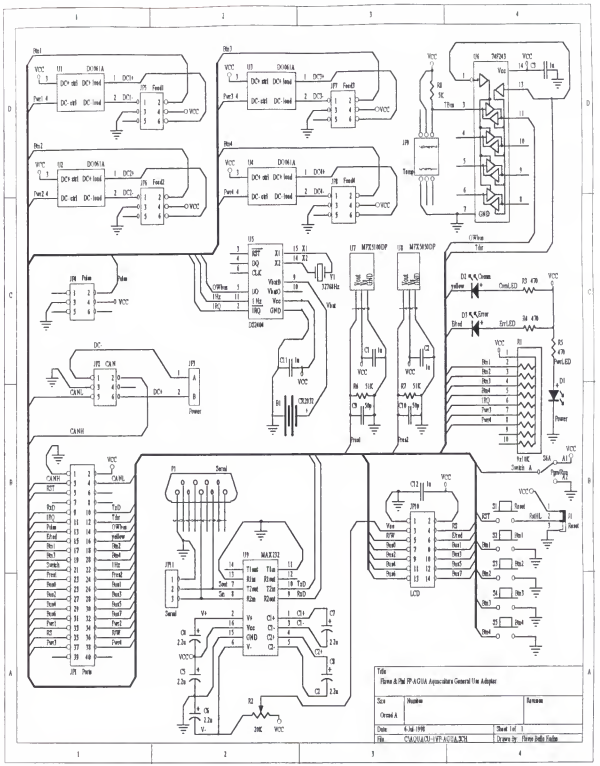


Figure 32. AGUA Daughterboard schematic.

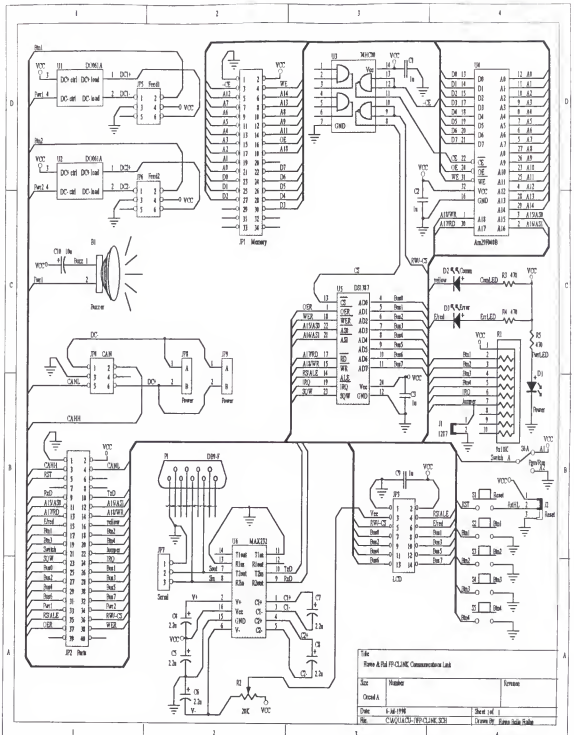


Figure 33. CLINK Daughterboard schematic.

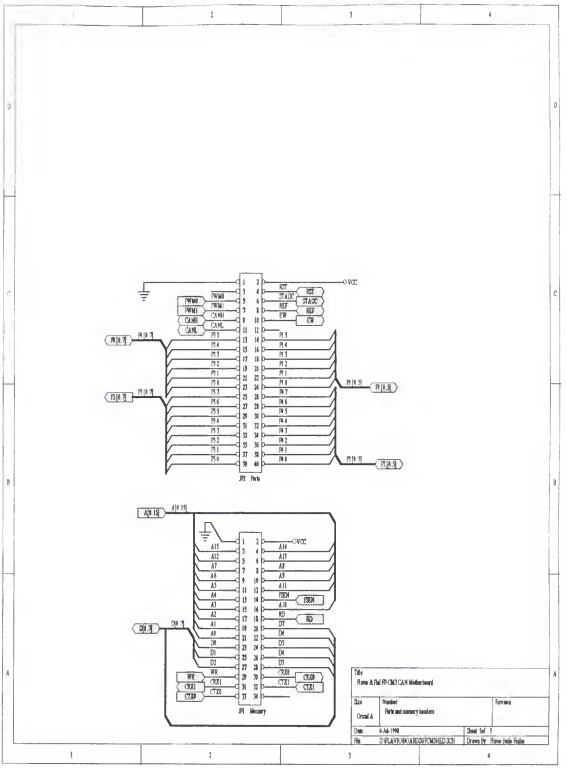


Figure 34.Header Schematic.

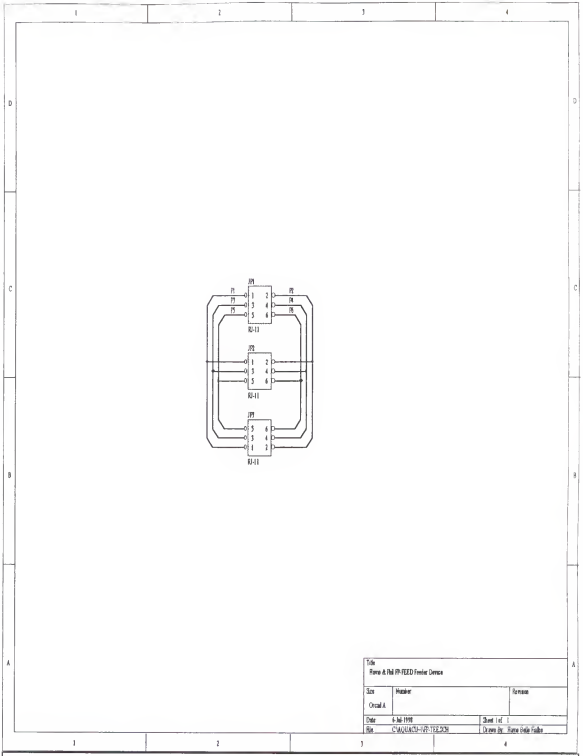


Figure 35. Tee Schematic.

APPENDIX B
SOFTWARE FLOWCHARTS

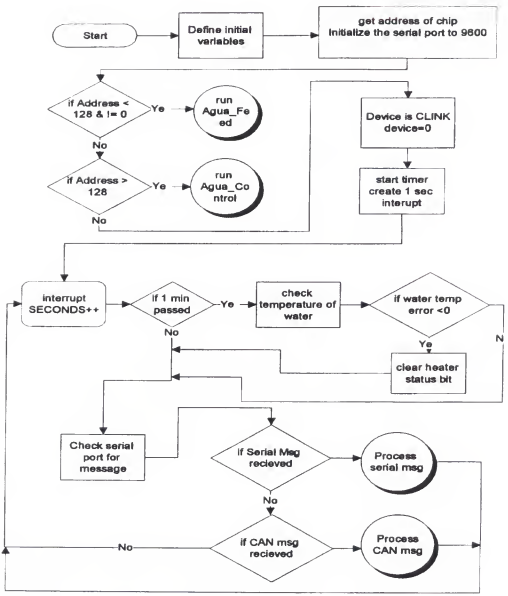


Figure 36. Motherboard initialization.

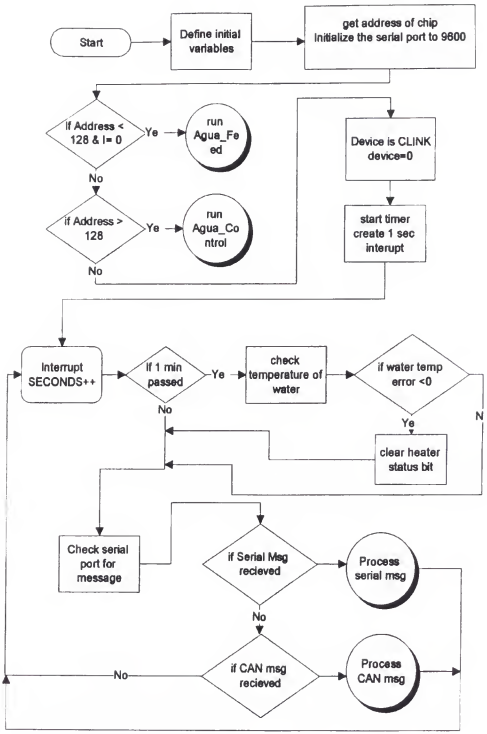


Figure 37. AGUA-feed flowchart.

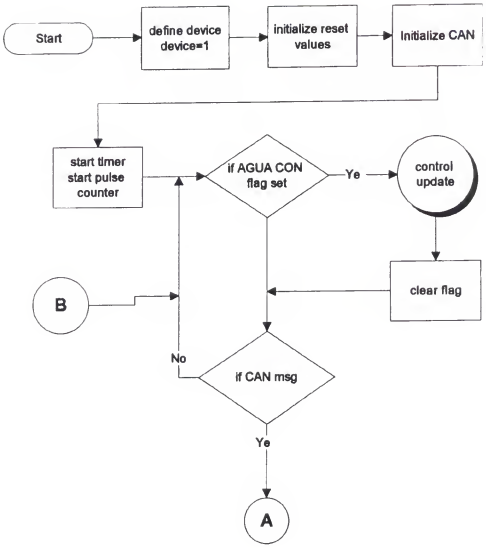


Figure 38. AGUA-con flowchart.

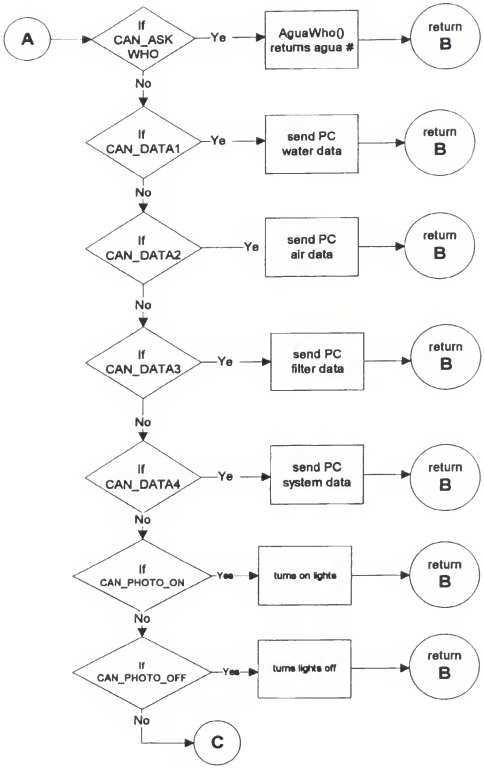


Figure 39. AGUA-con flowchart continued.

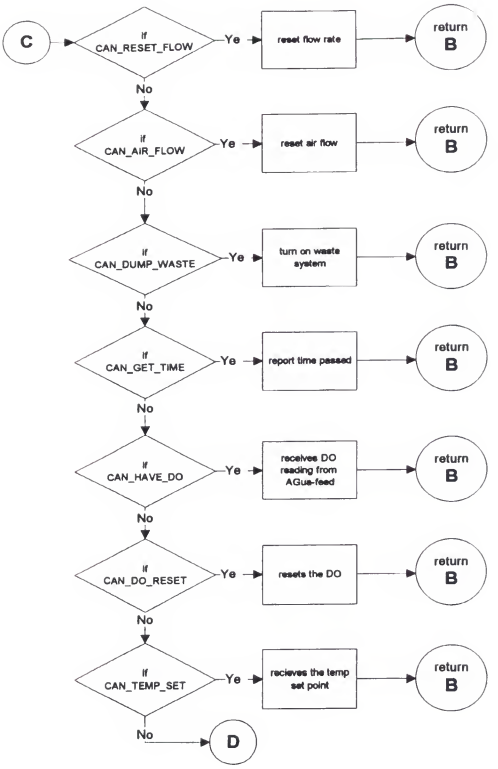


Figure 40. AGUA-con flowchart continued.

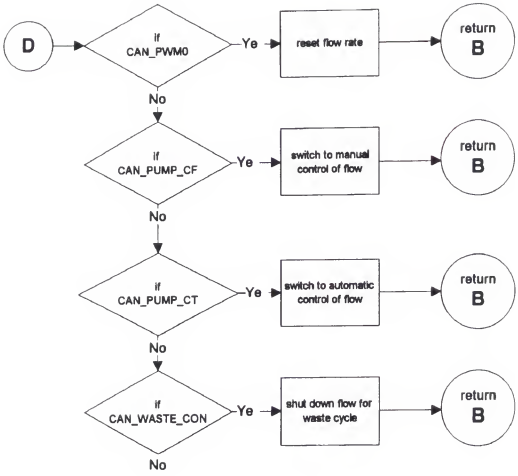


Figure 41. AGUA-con flowchart continued.

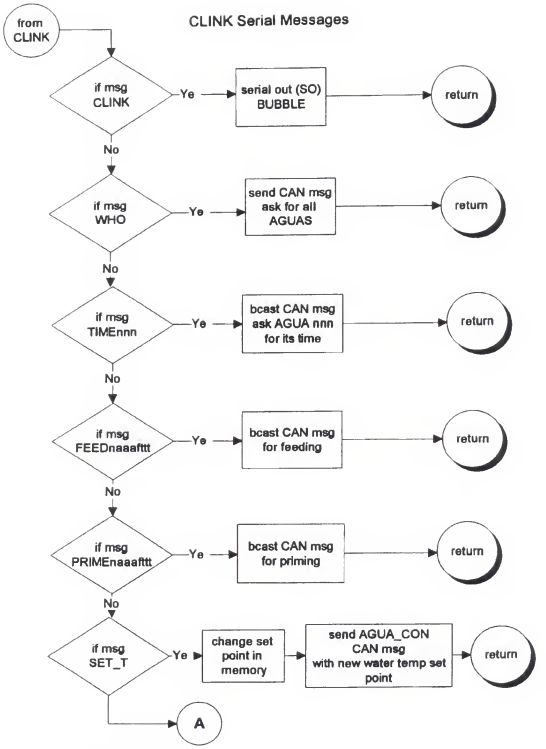


Figure 42. CLINK flowchart.

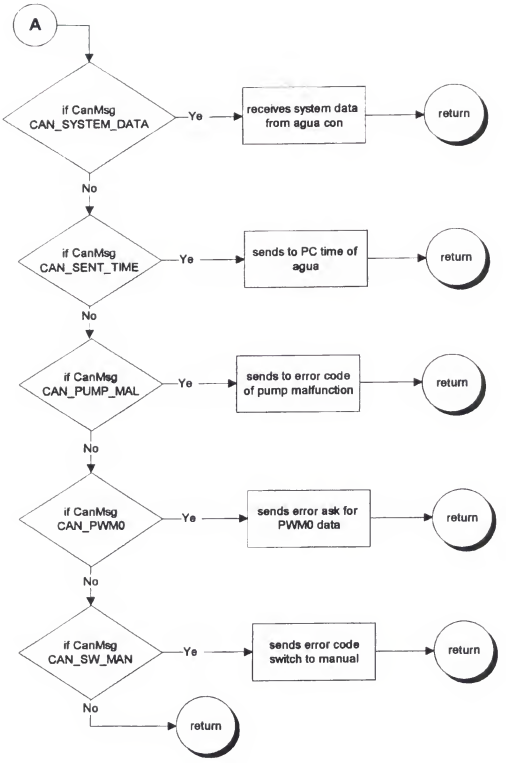


Figure 43.CLINK flowchart continued.

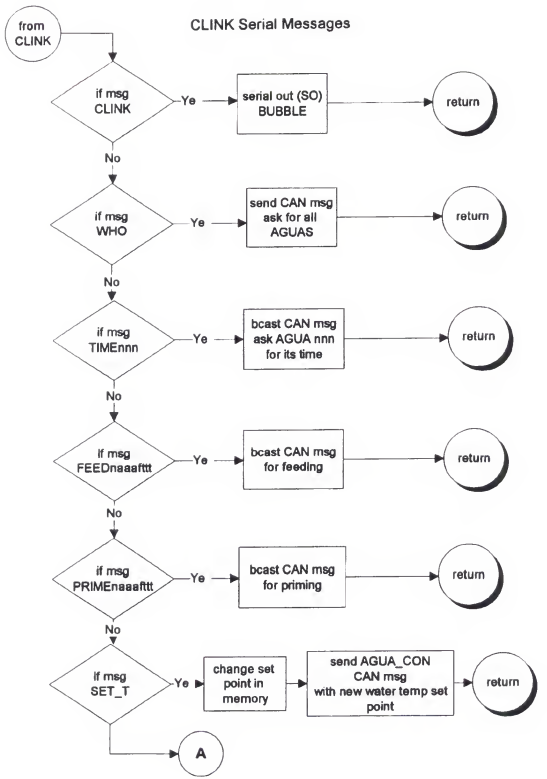


Figure 44. CLINK serial messages flowchart.

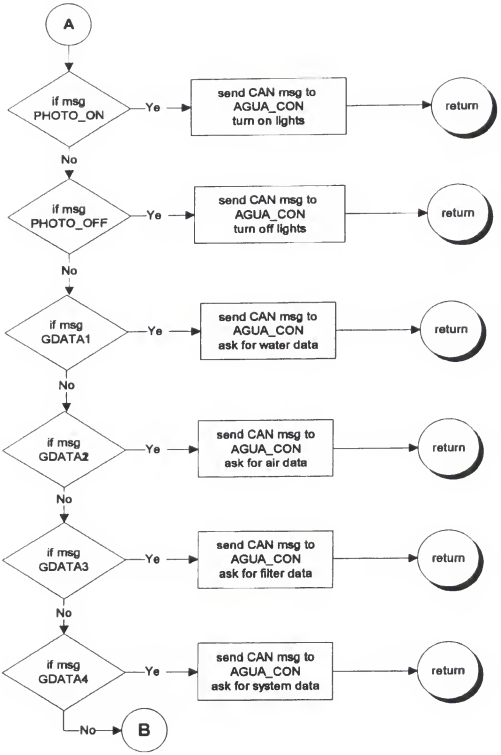


Figure 45. CLINK serial messages flowchart continued.

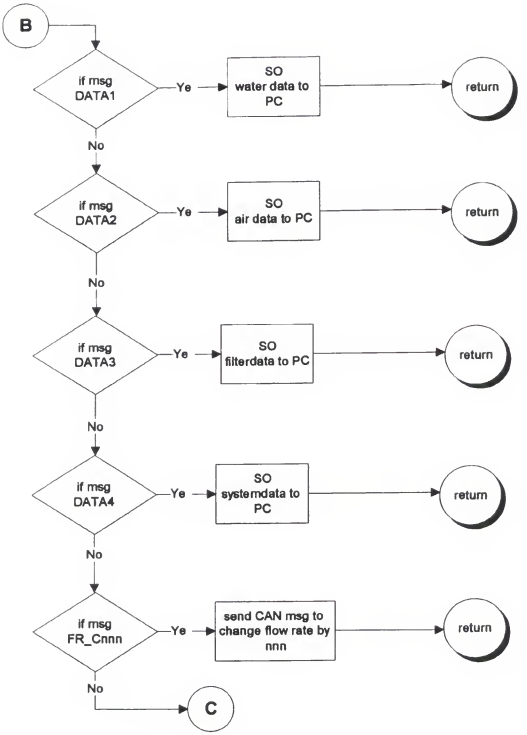


Figure 46. CLINK serial messages flowchart continued.

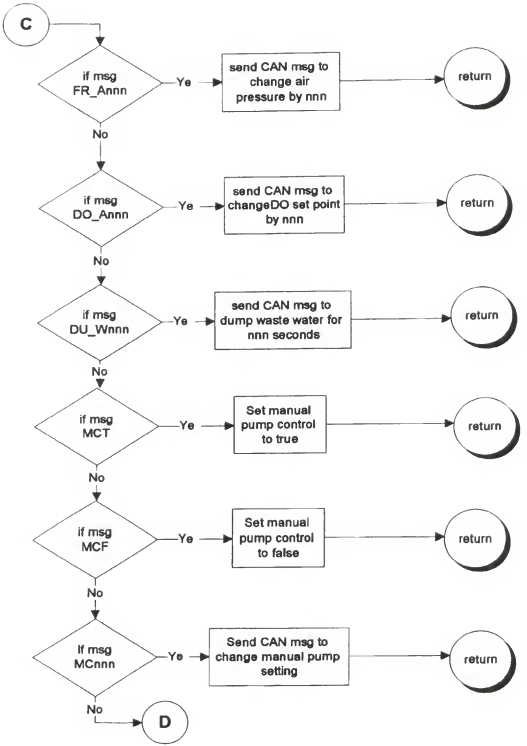


Figure 47. CLINK serial messages flowchart continued.

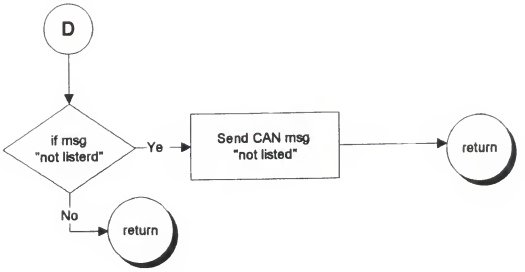


Figure 48. CLINK serial messages flowchart continued.

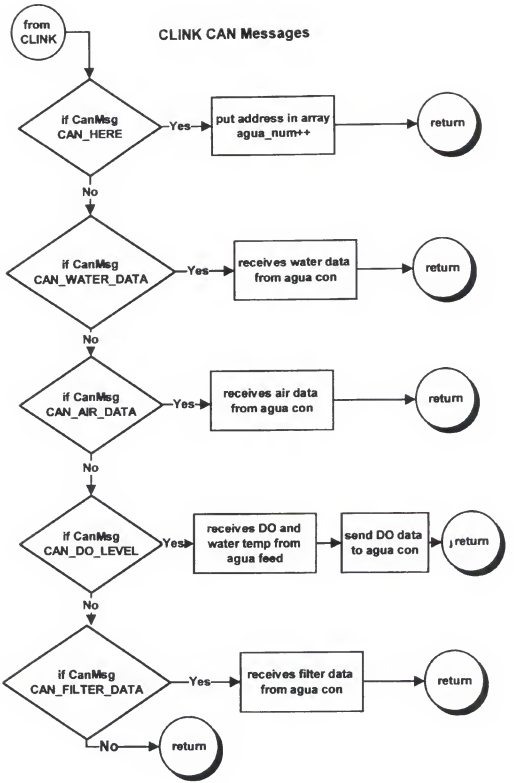


Figure 49. CLINK CAN messages.

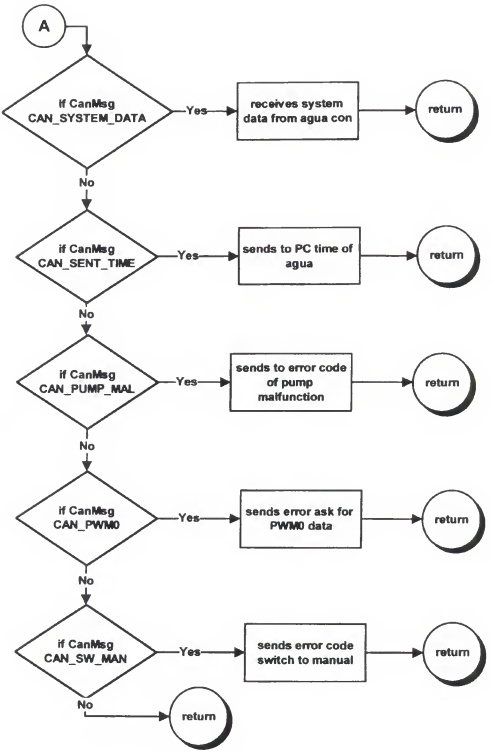


Figure 50. CLINK CAN messages continued.

APPENDIX C
MICROCONTROLLER CODE

```
//-------------------------------------------------------------------------------
// can_com.h
// interrupt based CAN interface for Philips P80C592 microcontroller
// Phil Fowler /FBF
// Dept. of Agric. & Bio. Eng., University of Florida
// 23/Feb/1997
//-------------------------------------------------------------------------------


// Uses interrupt 5
// Our CAN protocol:
// 1. Up to four (0-3) logical networks are allowed in the CAN
// 2. Each network has only one master and up to 255 slaves
// 3. All messages contain the network number and the target (master or slave)
// 4. A message to the master of a network contains the address of the sender
// 5. A message to a slave of a network contains the address of the slave
// 6. Messages to slave 0 are broadcast: all slaves in the network respond
// 7. Each device only receives messages from the network it belongs to
// 8. Any device can send any message to any network
// 9. Devices 240-255 cannot send messages to the master of network 3
// A message buffer has 10 bytes;
// the first 2 bytes are
// the 11-bit message identifier,
// the RTR bit (not used) and
// the message size;
// the message identifier defines
// the network,
// the target (master or slave) and
// the address of the device;
// thus, the first 2 bytes of the message are:
// [0]: nnta aaaa;
// [1]: aaar ssss
// nn = network (00 - 11)(0-3)
// t = target: defines the receiver of the message (1=master, 0=slave);
// when initializing the can, t should be 1 if the device is a master or
```

```
// 0 if it is a slave.
// aaaaaaaa = 8-bit address of the sending (t=1) or destination (t=0) device
// r = RTR bit
// ssss = message size
// When initializing the CAN, only the network and target are important; the
// acceptance mask os defined as 0001 111
enum
{
        CLINK=0,
   AGUA_CON=1,
   AGUA_FEED=2
};

enum
{
        CAN_BROADCAST = 0, // slave address for a broadcast message
        CAN_NETWORK = 0x40, // multiply this by the network number
        CAN_TARGET = 0x20 // multiply this by 0 (for a slave) or 1 (for a master)
};

enum
{
        CAN_ASKWHO=1,
        CAN_HERE=2,
   CAN_FEED=3,
        CAN_DATA1=4,
        CAN_DATA2=5,
   CAN_DATA3=6,
   CAN_DATA4=7,
   CAN_WATER_DATA=8,
   CAN_AIR_DATA=9,
   CAN_FILTER_DATA=10,
   CAN_SYSTEM_DATA=11,
   CAN_PHOTO_ON=12,
   CAN_PHOTO_OFF=13,
   CAN_RESET_FLOW=14,
   CAN_AIR_FLOW=15,
   CAN_DUMP_WASTE=16,
   CAN_GET_TIME=17,
   CAN_SENT_TIME=18,
   CAN_DO_LEVEL=19,
   CAN_TEMP_SET=20,
   CAN_HAVE_DO=21,
   CAN_DO_RESET=22,
   CAN_PWM0=23,
   CAN_PUMP_CT=24,
   CAN_PUMP_CF=25,
   CAN_PUMP_MAL=26,
   CAN_PWM0_DATA=27,
   CAN_SW_MAN=28,
   CAN_WASTE_CON=29
};
```

```
struct WData{
        unsigned char Water_Pump;    //PWM setting of water pump
   unsigned int FR_water_H;    //Flow meter reading high byte
   unsigned int FR_water_L;    //Flow meter reading low byte
   unsigned char FR_Set;       //flow rate setting high byte
   unsigned char Man_PWM0;     //manual setting for PWM0
   };

struct AData{
   unsigned char Air_Pump;      //PWM setting of air pump
   unsigned char A_Press_H;     //air pressure high byte
   unsigned char A_Press_L;     //air pressure low byte
   unsigned char A_Set;         //set point for air
   unsigned char DO;            //system DO (0-1v)(0-205,10bits)(0-14ppm)
   unsigned char DO_Set;        //DO set point
   };

struct FData{
        unsigned int  F_Press;       //differential filter pressure
   unsigned char F_Press_H;     //filter press high byte
   unsigned char F_Press_L;     //filter press low byte
   unsigned char Filter_C_set_H; //set filter control high byte
   unsigned char Filter_C_set_L; //set filter control low byte
   unsigned int  Filter_C_set;  //set point for diff filter
   };

struct SData{
   unsigned char Total_Feed;    //total feed delivered
   unsigned char L_Status;      // status of the lights
   unsigned char AC_update_t;   //time between readings for AGUA con
   unsigned char FD_update_t;   //time between readings for feeder con
   int           Water_Temp;    //system water temperature  degrees C
   int                          Air_Temp;      //air temperature degrees C
   int                          Water_Temp_Set; //water temperature set point
   unsigned char Heat_Status;   // 1-heater is on 0-heater is off
   };



void InitAddress (void);
        // Reads and stores the board address

void InitCan (unsigned char ACode, unsigned int BaudRate);
        // initialize the on-chip CAN registers
        // ACode = acceptance code register; BaudRate = baud rate / 100

void FreeCanIn (void);
        // releases the input buffer with the current message for further input

xdat unsigned char * CanIn (void);
        // returns a pointer to the next available message
        // or 0 if no messages are available

bit CanOutBusy (void);
```

```
        // returns the number of messages waiting in the CAN output buffer (0 or 1)

void CanOut (unsigned char * message);
        // transmits a message into the CAN; 'data' points to the start of the
        // message; if there is a message waiting in the queue, it is lost

#define TRUE 1
#define FALSE 0
#define FILTER_DIF 6
#define AIR_PRESS 7

void Agua_Control(void); //main loop for AGUA con
void Agua_Feed(void);    //main loop for AGUA feed
void Clink(void);        //main loop for clink
void AguaWho(void);
void WhoThere(void);
void Feeders(char *feed_code);
void Feed_Culture (unsigned char feeder,unsigned char time_on);
void setup_timer(void);
void setup_pulse_ctn(void);
unsigned int get_PULSE(void);
void reset_PULSE(void);
void Control_Update(void);
void Adjust_AIR(void);
void Dump_Waste(int);
void Button_Check(void);
unsigned int get_ADC(int);
void Send_data1(void);
void Send_data2(void);
void Send_data3(void);
void Send_data4(void);
void ClinkGetData1(void);
void ClinkGetData2(void);
void ClinkGetData3(void);
void ClinkGetData4(void);
void Photo_on(void);
void Photo_off(void);
void Reset_flow_rate(void);
void Reset_Air_rate(unsigned int);
void Reset_DO(unsigned char);
void D_Waste(unsigned int);
void Ask_can_time(unsigned char);
void Send_time(void);
void Send_DO(void);
void Agua_Feed_Update(void);
void Send_agua_con_DO(void);
void Adjust_DO(void);
int GetTemp(void);
void Reset_Temp(int);
void check_water_temp(void);
void UP_PWM0(void);
void UP_PUMP_CT(void);
void UP_PUMP_CF(void);
void Update_data(void);
```

```
void Send_Error(unsigned char);
void Set_Waste_C(void);
void dummy(void);

//-----------------------------------------------------------------------------
// a_temp.H
// Dallas DS1820 driver for 8051 compatible microcontroller
// Flavio Bello Fialho and Phil Fowler
// 20/Dec/1997
//-----------------------------------------------------------------------------
#ifndef a_temp_H
#define a_temp_H
#include <reg592.sfr>
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Hardware dependent definitions
//-----------------------------------------------------------------------------
// Port controlling the direction of information flow to temperature chips:
#define TEMP_DIR P3_3
//-----------------------------------------------------------------------------
// Ports where each temperature bus is connected (#undef the ones not used):
#define TEMP_BUS_0 P3_5

//-----------------------------------------------------------------------------
// Obs: Parasite power is not supported in this version of the driver
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Maximum wait time for a bus to be ready, ms (1..255):
#define MAXWAIT 250
//-----------------------------------------------------------------------------



//-----------------------------------------------------------------------------
// Constants for communication with the DS1820 chips
//-----------------------------------------------------------------------------
enum { IN, OUT };
enum {
        READ_ROM=0x33,
        MATCH_ROM=0x55,
        SKIP_ROM=0xCC,
        SEARCH_ROM=0xF0,
        ALARM_SEARCH=0xEC
};
enum {
        WRITE_SP=0x4E,
        READ_SP=0xBE,
        COPY_SP=0x48,
        CONVERT_T=0x44,
        RECALL_E2=0xB8,
        READ_POWER=0xB4
};
//-----------------------------------------------------------------------------
```

```
//-------------------------------------------------------------------------------
// Low level routines
//-------------------------------------------------------------------------------
// The following routine sends a reset pulse to the temperature sensor busses.
// ports = mask specifying to which busses the signal is to be sent (0..0xFF)
// returns: a mask specifying which busses responded to the reset signal
// * If a bus is shorted to Vcc, sensors will not be detected on that bus
// * If a bus is shorted to ground, PANIC!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
unsigned char TempResetPulse (unsigned char ports);
//-------------------------------------------------------------------------------
// The following routine writes one bit to the temperature sensor busses.
// ports = mask specifying to which busses the bit is to be sent (0..0xFF)
// wbit = bit to be written (0 or 1)
void TempWriteBit (unsigned char ports, bit wbit);
//-------------------------------------------------------------------------------
// The following routine reads one bit from the temperature sensor busses.
// ports = mask specifying from which busses the bit is to be read (0..0xFF)
// returns: bit read (0 or 1), ANDed between the selected busses
extern bit TempReadBit (unsigned char ports);
//-------------------------------------------------------------------------------

//-------------------------------------------------------------------------------
// Medium level routines
//-------------------------------------------------------------------------------
// The following routine writes one byte to the temperature sensor busses.
// ports = mask specifying to which busses the byte is to be sent (0..0xFF)
// wbyte = byte to be written; the bits are written from low to high
extern void TempWriteByte (unsigned char ports, unsigned char wbyte);
//-------------------------------------------------------------------------------
// The following routine reads one byte from the temperature sensor busses.
// ports = mask specifying to which busses the byte is to be sent (0..0xFF)
// returns: byte read, ANDed between busses; bits are read from low to high
extern unsigned char TempReadByte (unsigned char ports);
//-------------------------------------------------------------------------------
#endif

// can_com.c
// interrupt based CAN interface for Philips P80C592 microcontroller to test can
// Phil Fowler and FBF
// Dept. of Agric. & Bio. Eng., University of Florida
// 4/03/98
//-----------------------------things to do-------------------------------------
// set up fail safe for faulty DO probe
// get the temperature to increase at no greater than 1 degree per hour
//-------------------------------------------------------------------------------

#include <reg592.sfr>
#include <string.h>
#include <stdlib.h>
#include "hardware.h"
#include "can_com.h"
#include "serial.h"
#include "FP_time.h"
```

```
#define CAN_BUFFER_SIZE 10
#define CAN_BAUD 96

unsigned char mess_size;
data unsigned char usDelay_i;
data int msDelay_i;
volatile bit CanMessageIn = 0;
volatile bit CanOutFree = 1;
static volatile xdat char InBuffer [2] [CAN_BUFFER_SIZE];
static volatile char MessageIn [2] = { 0, 0 };
static volatile char ActiveInBuf = 0;
static xdat char OutBuffer [CAN_BUFFER_SIZE];
static volatile bit CanOutIdle = 1;
char * SerialMsg; // points to a message in the serial port buffer
_bit ACFLAG=0;
_bit FDFLAG=0;
_bit PUMP_C=1;  //set pump control mode to automatic
_bit MAN_PUMP_C=1; //sets the manual program override for pump control to automatic
_bit DIFF_CONTROL=0; //sets differential flag to false (off)
_bit WASTE_C=0;  //waste control flage if 1 on 0 off
unsigned long T2tick=0,QUARTER_SECONDS=0,AGUA_FEED_SECONDS=0;
unsigned long SECONDS=0,MIN=0,SEC_COUNT=0,PULSE_HIGH,PULSE_LOW,fil_tot=0;
unsigned int tick_count=0,air_c,air_ave;
unsigned int tick_cnt2=0;
unsigned int FD_update=20;  //update time for process control of feeder  20 SECONDS
unsigned int PULSE_CNT,PULSE_CNT_METER,AVE_PWM0=0,PWM0_TOT;
unsigned long filcount=0;
unsigned int P_set=0; //pressure set point for pressure control of PWM of flowrate

int Watertemp;
long dump_fin,feed_fin;
xdat unsigned char * CanMsg; // last CAN message received
unsigned char agua_num=0;//number of AGUAS
unsigned char device,time_slice,pump_check=0;
unsigned char Address; // board address
volatile bit CanMessageIn; // 1 if a CAN message is available; 0 if not
volatile bit CanOutFree; // 1 if the CAN output buffer is free; 0 if not

struct WData W_data;  //holds the data for water
struct AData A_data;  //holds the data for air
struct FData F_data;  //holds the data for filter
struct SData S_data;  //holds the data for system

void main (void)
{
    //initialize all the data structures
   W_data.Water_Pump=100;    //initial water pump PWM setting
   W_data.FR_water_H=0;      //no flow at start
   W_data.FR_water_L=0;
   W_data.FR_Set=40;
   W_data.Man_PWM0=128;      //manual set to 50%;

   A_data.Air_Pump=18;       //PWM setting of air pump
   A_data.A_Press_H=0;       //air pressure high byte
```

```
   A_data.A_Press_L=0;     //air pressure low byte
   A_data.A_Set=100;       //set point for air
   A_data.DO=0;            //system DO (0-1v)(0-205,10bits)(0-14ppm)
   A_data.DO_Set=50;       //DO set point

   F_data.F_Press=0;      //filter press
   F_data.F_Press_H=0;    //filter press high byte
   F_data.F_Press_L=0;    //filter press low byte
   F_data.Filter_C_set_H=0; //set filter control high byte
   F_data.Filter_C_set_L=0; //set filter control low byte
   F_data.Filter_C_set;  //set point for diff filter

   S_data.Total_Feed=0;    //total feed delivered
   S_data.L_Status=0;      // status of the lights
   S_data.AC_update_t=3;   //time between readings for AGUA con  must adjust flowrate to match
update!!!!  3 SECONDS
   S_data.FD_update_t=20;  //time between readings for feeder con
   S_data.Water_Temp=0;    //system temperature
   S_data.Water_Temp_Set=22;//set point for water temperature
   S_data.Air_Temp=0;                  // air temperature degrees C
   S_data.Heat_Status=0;   ///default heater is off

         InitAddress();    //gets the address of the chip
         InitSerial(96);
         SerialOutString ("can com 06\n");
         SerialOutHex(Address);
         SerialOutString("\n");

         if (Address<128 && Address!=0)    //all AGUAS with addresses less than 128 are feeders
          {
           SerialOutString ("AGUA_FEED\n");
           Agua_Feed();
          }
         else if (Address>=128)  //all AGUAS with addresses greater than 128 are control units
          {
           SerialOutString ("AGUA_CON\n");
           Agua_Control();
          }
         else
          {
           SerialOutString ("Clink\n");
           Clink();
          }
}

void Clink (void)
{
         // Main program loop if board is a Clink

         unsigned char i,j=0;
   unsigned char agua[256];  //255 possible AGUAS
   device=CLINK;
   setup_timer(); //initialize timer
```

```
for(i=0;i<=255;i++)agua[i]=0; //clear the array

      // Initialize the CAN for CLINK (network=0, target=1 (master), speed=9600)
      InitCan(0 * CAN_NETWORK + 1 * CAN_TARGET, CAN_BAUD);


while(1)          // Repeat the following loop forever:
{
 if(SECONDS%60==0)check_water_temp();//check water temp every min and adjust if needed
                        //change this later(longer time)
                // Test if there is a message in the serial port input buffer and
                // process it, if there is one:
        SerialMsg = SerialIn (); // get address of the next serial port message
        if (SerialMsg)
         {      // there is a message in the serial port, process it
                 if (! strcmp (SerialMsg, "CLINK?"))SerialOutString ("BUBBLE!\n");
           else
                 if (! strcmp (SerialMsg, "WHO?"))
    {
      WhoThere(); //finds out who is out there by address
      SerialOutHex (agua_num);
      SerialOutString (" reporting AGUAS\n");
     for(i=0;i<=5;i++)
     {
     SerialOutString ("Agua  ");
     SerialOutHex (agua[i]);
     SerialOutString ("\n");
     }
    }
    else
    if (! strncmp (SerialMsg, "TIME",4))Ask_can_time(atoi(SerialMsg +4));
    else                        //enter the CAN address
    if (! strncmp (SerialMsg, "FEED",4)) Feeders(SerialMsg+4);
                        //which network,which AGUA, which feeder, and for how long naaafttt
      //n   network
                        //aaa address of feeder (1-127)
                        //f  feeder number (0-3)
                        //ttt time in sec (0-999)
    else
    if (! strncmp (SerialMsg, "PRIME",5))Feeders(SerialMsg+5);
    //this may need to change to keep from logging from priming
      //which network,which AGUA, which feeder, and for how long naaafttt
      //n   network
                        //aaa address of feeder (1-127)
                        //f  feeder number (0-3)
                        //ttt time in 1/4 sec (0-999)
                 else
    if (! strncmp (SerialMsg, "SET_T",5))
                 {
          S_data.Water_Temp_Set=atoi(SerialMsg+5);
              Reset_Temp(S_data.Water_Temp_Set);
        }
    else
    if (! strcmp (SerialMsg, "PHOTO_ON?")) Photo_on();
```

```
else
if (! strcmp (SerialMsg, "PHOTO_OFF?")) Photo_off();
else
if (! strcmp (SerialMsg, "GDATA1"))ClinkGetData1(); // get the data
else
if (!strncmp (SerialMsg, "GDATA2",6))ClinkGetData2(); // msg from PC to get the data
else
if (! strcmp (SerialMsg, "GDATA3"))ClinkGetData3(); // get the data
else
if (! strcmp (SerialMsg, "GDATA4"))ClinkGetData4(); // get the data from AGUA_con
else
                if (! strcmp (SerialMsg, "DATA1"))
                {  //sends data for water to PC

   SerialOutHex (0x4);
   SerialOutHex (W_data.Water_Pump);   //water pump PWM0
   SerialOutHex (W_data.FR_water_H);
   SerialOutHex (W_data.FR_water_L);
   SerialOutHex (W_data.FR_Set);
   SerialOutHex (W_data.Man_PWM0);     //manual pump setting
   SerialOutHex (0x0);
   SerialOutHex (0x0);
   SerialOutHex (0x0);
   SerialOutString("\n");

                }
                else
if (! strcmp (SerialMsg, "DATA2"))
                {  //sends data of air  to PC
                SerialOutHex (0x5);
   SerialOutHex (A_data.Air_Pump);    //datab1
   SerialOutHex (A_data.A_Press_H);   //datab2
   SerialOutHex (A_data.A_Press_L);   //datab3
   SerialOutHex (A_data.A_Set);       //datab4
   SerialOutHex (A_data.DO);          //datab5
   SerialOutHex (A_data.DO_Set);      //datab6
   SerialOutHex (0x0);
   SerialOutHex (0x0);
   SerialOutString("\n");
                }
                else
if (! strcmp (SerialMsg, "DATA3"))
                {  //sends data for filter to PC
                SerialOutHex (0x6);
   SerialOutHex (F_data.F_Press_H);
   SerialOutHex (F_data.F_Press_L);
   SerialOutHex (F_data.Filter_C_set_H);
   SerialOutHex (F_data.Filter_C_set_L);
   SerialOutHex (0x0);
   SerialOutHex (0x0);
   SerialOutHex (0x0);
   SerialOutHex (0x0);
   SerialOutHex (0x0);
   SerialOutHex (0x0);
```

```
  SerialOutString("\n");
            }
            else
if (! strcmp (SerialMsg, "DATA4"))
            {   //sends data for system to PC
            SerialOutHex (0x7);            //data_code
  SerialOutHex (S_data.Total_Feed);  //databl
  SerialOutHex (S_data.L_Status);    //databl2
  SerialOutHex (S_data.AC_update_t); //AGUA con update time  //databl3
  SerialOutHex (S_data.FD_update_t); //AGUA feed update time  //databl4
  SerialOutHex (S_data.Water_Temp);  //water temperature in C //databl5
  SerialOutHex (S_data.Air_Temp);    //air temperature      //databl6
  SerialOutHex (S_data.Water_Temp_Set);//water temp set point //databl7
  SerialOutHex (S_data.Heat_Status);  //heater status bit   //databl8
  SerialOutString("\n");
            }
            else
if (!strncmp (SerialMsg, "FR_C",4)) //changes the flow rate setting
{
   W_data.FR_Set=atoi(SerialMsg +4);  //change value of flow rate setting
  Reset_flow_rate();              //send new value to AGUA-con

}
else
if (!strcmp (SerialMsg, "WC_C")) //changes the flow rate setting
{
  Set_Waste_C();                  //send message to disable control functions
}
else
if (!strncmp (SerialMsg, "FR_A",4))
{
   A_data.A_Set=(atoi(SerialMsg +4));
  Reset_Air_rate(A_data.A_Set);
}
else
if (!strncmp (SerialMsg, "DO_A",4))Reset_DO((atoi(SerialMsg +4)));
else
if (!strncmp (SerialMsg, "DU_W",4))D_Waste(atoi(SerialMsg +4));
else
if (! strcmp (SerialMsg, "MCT"))
{
UP_PUMP_CT(); // set the manual pump control flag
}
else
if (! strcmp (SerialMsg, "MCF"))UP_PUMP_CF(); // clear the manual pump control flag
else
if (!strncmp (SerialMsg, "MC",2))
 {  //manual control setting
   W_data.Man_PWM0=2560/(1000/(atoi(SerialMsg +2)));
   UP_PWM0();
 }
else
            {
             SerialOutString (SerialMsg);
```

```
            SerialOutString (" to you too!\n");
            }
            FreeSerialIn ();
    }
        //set up read CAN messages routine

if (CanMessageIn)           // if there is a message in the CAN, process it
{

  CanMsg = CanIn (); //point to the begining of message
  mess_size=(CanMsg[1]&0xF)+2; //number of bytes of message + identifier(2 bytes)
  for(i=0;i<=mess_size-1;i++) //starts with 0 so reduce mess_size by 1
  {

  }
  switch(CanMsg[2])
  {
    case    CAN_HERE: // if there is an agau, count it, store its address
            agua[agua_num] = (CanMsg [0] << 3) + (CanMsg [1] >> 5); // get slave address
            agua_num++;         //increment array
                    break;

    case    CAN_WATER_DATA: //recieves water data from AGUA con

            W_data.Water_Pump=CanMsg [3];
            W_data.FR_water_H=CanMsg [4];
            W_data.FR_water_L=CanMsg [5];
        W_data.FR_Set=CanMsg [6];
            break;

    case    CAN_AIR_DATA: //recieves air data from AGUA con

        A_data.Air_Pump=CanMsg [3];
            A_data.A_Press_H=CanMsg [4];
            A_data.A_Press_L=CanMsg [5];
            A_data.A_Set=CanMsg [6];
        A_data.DO=CanMsg [7];
        A_data.DO_Set=CanMsg [8];
            break;

    case  CAN_DO_LEVEL:   //receives DO level and water temp from AGUA feed
            A_data.DO=CanMsg[3];
        S_data.Water_Temp=CanMsg[4];
        Send_agua_con_DO(); //send DO level to AGUA con for control
        break;

    case    CAN_FILTER_DATA:  //recieves filter data from AGUA con
       F_data.F_Press_H=CanMsg [3];
                F_data.F_Press_L=CanMsg [4];
            F_data.Filter_C_set_H=CanMsg [5];
                F_data.Filter_C_set_L=CanMsg [6];
            break;

    case CAN_SYSTEM_DATA:  //recieves system data from AGUA con ????
```

```
                    S_data.Total_Feed=CanMsg [3];
                    S_data.L_Status=CanMsg [4];
                    S_data.AC_update_t=CanMsg [5];
            S_data.FD_update_t=CanMsg [6];
            S_data.Air_Temp=CanMsg[7];
                    break;

        case CAN_SENT_TIME:
                    SerialOutHex(CanMsg[6]);        // 0  BYTE
            SerialOutHex(CanMsg[5]);    // 1  BYTE
            SerialOutHex(CanMsg[4]);    // 2  BYTE
            SerialOutHex(CanMsg[3]);    // 3  BYTE
                    SerialOutString ("\n");
            break;

        case CAN_PUMP_MAL:
            SerialOutHex (0x8);         //error_code
            SerialOutHex (0x1);         //pump malfunction
            break;

        case CAN_PWM0_DATA:
            SerialOutHex (0x8);         //error_code
            SerialOutHex (0x2);         //send PWMO data
            break;

        case CAN_SW_MAN:
            SerialOutHex (0x8);         //error_code
            SerialOutHex (0x3);         //send PWMO data
            break;
        }
    }
            FreeCanIn (); // free the input buffer to get other messages
 }
}

void Agua_Control(void)   //code for an AGUA which contols the system
{

  device=AGUA_CON;   //type of device
       ERROR_LED = COMM_LED = 1;
       PULSE=1;
       PWMP=0x14;    //frequency on reset
       PWM0=0x7F;    //water pump on reset(7F)
       air_c=PWM1=0x12;       //air pump on reset
  PHOTO_4=1;       //light off at reset
  S_data.L_Status=0; //light off at reset
  A_data.DO_Set=50;  //DO set point on reset
  PUMP_C=1;        //put in automatic mode for pump control

       // Initialize the CAN for VACA (network=0, target=0 (slave), speed=9600)
       InitCan(0 * CAN_NETWORK + 0 * CAN_TARGET, CAN_BAUD);
       setup_timer();
       setup_pulse_ctn();
```

```
while(1)          // Repeat the following loop forever
{
        if(ACFLAG)      //measures flow rate every update secs
{

Control_Update();         //checks conditons of processes and resets control parameters
ACFLAG=0;
}

// First, test if there is a message wating in the CAN input buffer and
// process it, if there is one
        if (CanMessageIn)                   // if there is a message in the CAN, process it
        {
                CanMsg = CanIn(); // get the address of the next CAN message
                switch (CanMsg[2])
                {
case CAN_ASKWHO:
        AguaWho (); // message = Who is out there?
                break;
  case CAN_DATA1: //send back the water data from the system
    Send_data1();
        break;

  case CAN_DATA2: //send back to clink the air data from the system
    Send_data2();
        break;

  case CAN_DATA3: //send back the filter data from the system
    Send_data3();
        break;

  case CAN_DATA4: //send back the system data from the system
    Send_data4();
        break;

  case CAN_PHOTO_ON:  //turns the lights on
    PHOTO_4=0;
                        S_data.L_Status=1; //the lights are on(needs non-volitile memory)
    break;

  case CAN_PHOTO_OFF:   //turns the lights off
    PHOTO_4=1;
                        S_data.L_Status=0; //the lights are off(needs non-volitile memory)
    break;

  case CAN_RESET_FLOW:      //change the set point of the flow rate
        W_data.FR_Set=CanMsg[3];
    PWM0=100;           //starts the pump
    break;

  case CAN_AIR_FLOW:      //change the air flow set point
        A_data.A_Set=CanMsg[3];
    break;
```

```
            case CAN_DUMP_WASTE:          //dump the waste from the system
                    Dump_Waste(CanMsg[3]);
                break;

            case CAN_GET_TIME:          //gives the seconds passed on AGUA
                Send_time();
                break;

            case CAN_HAVE_DO:
                    A_data.DO=CanMsg[3];    //recieves DO reading from AGUA feeder
                break;

            case CAN_DO_RESET:
                A_data.DO_Set=CanMsg[3];      //reset the DO set point
                break;

            case CAN_TEMP_SET:
                    S_data.Water_Temp_Set=CanMsg[3];
                break;

            case CAN_PWM0:
                    W_data.Water_Pump=W_data.Man_PWM0=CanMsg[3];
                PWM0=W_data.Man_PWM0;
                break;

            case CAN_PUMP_CF:
                    PUMP_C=0;     //switch to manual control of water pump
                break;

            case CAN_PUMP_CT:
                PUMP_C=1;     //switch to automatic control of water pump
                break;

            case CAN_WASTE_CON:
                    WASTE_C=1;      //turn on waste control flag (disable control functions)
                PWM0=0;         //turn pump off
                break;

                        }
                    FreeCanIn (); // free the input buffer to get other messages

                    }

            }
}

void Agua_Feed(void)    //code for an AGUA which feeds the system
{
   device=AGUA_FEED;
        ERROR_LED = COMM_LED = 1;  //turns em off
        // Initialize the CAN for VACA (network=0, target=0 (slave), speed=9600)
        InitCan(0 * CAN_NETWORK + 0 * CAN_TARGET, CAN_BAUD);
        setup_timer();   //set up system timer for seconds count
```

```
        while(1)            // Repeat the following loop forever
        {
                if(FDFLAG)      //measures every FD_update secs (20 sec)
        {   //checks conditons of processes and resets control parameters
        Agua_Feed_Update();   //just gets the sytem DO
        Send_DO();
        FDFLAG=0;           //reset flag

        }

         // First, test if there is a message wating in the CAN input buffer and
         // process it, if there is one

                if (CanMessageIn)                   // if there is a message in the CAN, process it
                {
                        CanMsg = CanIn(); // get the address of the next CAN message
                        switch (CanMsg[2])
                        {
                        case CAN_ASKWHO:
                    SerialOutString("AGUA HERE\n");
                SerialOutHex(Address);
                AguaWho (); // message = Who is out there?
                break;

                        case CAN_FEED: // message = feed the culture.
                    SerialOutString("Will feed em\n");
                Feed_Culture (CanMsg [3],CanMsg[4]);
                                        break;

            case CAN_GET_TIME:
                Send_time();
                break;

                        }
                    FreeCanIn (); // free the input buffer to get other messages
                    }
         }
}

// ---------------------CAN routines-----------------------------------------
void AguaWho(void) // AGUA received a message asking who is online
            //and sends back a response
{       // message to output to the CAN
        // network=0, target=1 (master), slave=board address, size=1:
        unsigned char OutMessage [10];
        OutMessage [0] = 0 * CAN_NETWORK + 1 * CAN_TARGET + (Address >> 3);
        OutMessage [1] = (Address << 5) + 0x01;
        OutMessage [2] = CAN_HERE; // message = I am here!
        CanOut (OutMessage); // output the message to the CAN
}

void WhoThere(void)
{   //finds all the AGUAS
        unsigned char OutMessage [10];
```

```
    agua_num=0;//set the number of AGUAS to zero
          OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (CAN_BROADCAST >> 3);
                  // network = 0, target = 0 (slave), slave = BROADCAST
          OutMessage [1] = (CAN_BROADCAST << 5) + 0x01;
                  // slave = BROADCAST, message size = 1
          OutMessage [2] = CAN_ASKWHO; // message = are you there.
          CanOut (OutMessage); // output the message to the CAN

}

void Ask_can_time(unsigned char agua_add)
{  //sends a message to agua_add asking for its time
   unsigned char OutMessage [10];

   OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (agua_add >> 3);
                  // network = 0, target = 0 (slave), slave = BROADCAST
   OutMessage [1] = (agua_add << 5) + 0x01;  //one data byte
   OutMessage [2] = CAN_GET_TIME; // message = get time from an AGUA
   CanOut (OutMessage); // output the message to the CAN
}

void Send_time(void)
{ //is a request for the AGUAS time
 unsigned char OutMessage [10],SEC_B0=0,SEC_B1=0,SEC_B2=0,SEC_B3=0;

    SEC_B3=(SECONDS>>24)&0xFF;    //convert to bytes
    SEC_B2=(SECONDS>>16)&0xFF;
    SEC_B1=(SECONDS>>8)&0xFF;
    SEC_B0=(SECONDS&0xFF);

    OutMessage [0] = 0 * CAN_NETWORK + 1 * CAN_TARGET + (Address >> 3);
    OutMessage [1] = (Address << 5) + 0x05;
    OutMessage [2] = CAN_SENT_TIME; // message = get time from an AGUA
    OutMessage [3] = SEC_B0;       //low byte
    OutMessage [4] = SEC_B1;
    OutMessage [5] = SEC_B2;
    OutMessage [6] = SEC_B3;       //high byte
    CanOut (OutMessage); // output the message to the CAN
}

void Send_data1(void)
{  //sends water data from AGUA-con to CLINK
 unsigned char OutMessage [10];
    OutMessage [0] = 0 * CAN_NETWORK + 1 * CAN_TARGET + (Address >> 3);  //message to master
          OutMessage [1] = (Address << 5) + 0x05; //message and 4 bytes of data
          OutMessage [2] = CAN_WATER_DATA; // message = the data of water flow
    OutMessage [3] = W_data.Water_Pump;    //PWM0 value
    OutMessage [4] = W_data.FR_water_H;    //Flow rate high byte
    OutMessage [5] = W_data.FR_water_L;    //Flow rate low byte
    OutMessage [6] = W_data.FR_Set;      //flow rate set point high byte
          CanOut (OutMessage); // output the message to the CAN
 }

void Send_data2(void)     //send clink air data from AGUA con
```

```
{
unsigned char OutMessage [10];
  OutMessage [0] = 0 * CAN_NETWORK + 1 * CAN_TARGET + (Address >> 3);  //message to master
        OutMessage [1] = (Address << 5) + 0x07; //message and 7 bytes of data
        OutMessage [2] = CAN_AIR_DATA; // message = the data of air flow
  OutMessage [3] = A_data.Air_Pump;        //PWM setting of air pump
  OutMessage [4] = A_data.A_Press_H;       //air pressure high byte
  OutMessage [5] = A_data.A_Press_L;       //air pressure low byte
  OutMessage [6] = A_data.A_Set;         //set point for air high byte
  OutMessage [7] = A_data.DO;                 //system DO
  OutMessage [8] = A_data.DO_Set;                    //system DO set point
        CanOut (OutMessage); // output the message to the CAN
}

void Send_data3(void)   //send filter data from AGUA con to clink
{
unsigned char OutMessage [10];

  OutMessage [0] = 0 * CAN_NETWORK + 1 * CAN_TARGET + (Address >> 3);  //message to master
        OutMessage [1] = (Address << 5) + 0x05; //message and 4 bytes of data
        OutMessage [2] = CAN_FILTER_DATA; // message = the data of filter flow
  OutMessage [3] = F_data.F_Press_H;       //filter press high byte
  OutMessage [4] = F_data.F_Press_L;       //filter press low byte
  OutMessage [5] = F_data.Filter_C_set_H;   //set point filter control high byte
  OutMessage [6] = F_data.Filter_C_set_L;   //set point filter control low byte
        CanOut (OutMessage); // output the message to the CAN
}

void Send_data4(void)    //sends various system data  to Clink
{                 //from AGUA con
unsigned char OutMessage [10];

  OutMessage [0] = 0 * CAN_NETWORK + 1 * CAN_TARGET + (Address >> 3);  //message to master
        OutMessage [1] = (Address << 5) + 0x06; //message and 4 bytes of data
        OutMessage [2] = CAN_SYSTEM_DATA; // message = the data of system
  OutMessage [3] = S_data.Total_Feed;     //total feed delivered
  OutMessage [4] = S_data.L_Status;       // status of the lights
  OutMessage [5] = S_data.AC_update_t;       //time between readings fo AGUA con
  OutMessage [6] = S_data.FD_update_t;       //time between readings fo feeder con
  OutMessage [7] = S_data.Air_Temp;      //temperature of the air
        CanOut (OutMessage); // output the message to the CAN

}

void ClinkGetData1(void)
{
  unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                // network = 0, target = 0 (slave), slave = BROADCAST
  OutMessage [1] = (Address << 5) + 0x01;  //one data byte
  OutMessage [2] = CAN_DATA1; // message = send water data
  CanOut (OutMessage); // output the message to the CAN
```

```
}

void ClinkGetData2()
{  //sends message to AGUA con to send the air data
   unsigned char OutMessage [10];
   Address=0x81; //hard code address for testing
   OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                    // network = 0, target = 0 (slave), slave = address
   OutMessage [1] = (Address << 5) + 0x01;  //one data byte
   OutMessage [2] = CAN_DATA2; // message = send air data
   CanOut (OutMessage); // output the message to the CAN

}

void ClinkGetData3(void)
{  //sent to AGUA-con to ask for filter data
   unsigned char OutMessage [10];
   Address=0x81; //hard code address for testing
   OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
   OutMessage [1] = (Address << 5) + 0x01;  //one data byte
   OutMessage [2] = CAN_DATA3; // message = send filter data
   CanOut (OutMessage); // output the message to the CAN

}

void ClinkGetData4(void)
{
   unsigned char OutMessage [10];
   Address=0x81; //hard code address for testing
   OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                    // network = 0, target = 0 (slave), slave = BROADCAST
   OutMessage [1] = (Address << 5) + 0x01;  //one data byte
   OutMessage [2] = CAN_DATA4; // message = send system data
   CanOut (OutMessage); // output the message to the CAN

}

void Reset_flow_rate()
{   //message from clink to AGUA_con to change the flow rate set point
         unsigned char OutMessage [10];
   Address=0x81; //hard code address for testing
   OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                    // network = 0, target = 0 (slave), slave = BROADCAST
   OutMessage [1] = (Address << 5) + 0x02;  //two data bytes
   OutMessage [2] = CAN_RESET_FLOW; // message = send system data
   OutMessage [3] = W_data.FR_Set;
   CanOut (OutMessage); // output the message to the CAN

}

void Set_Waste_C(void)
{  //message from clink to AGUA_con to change the flow rate set point
         unsigned char OutMessage [10];
   Address=0x81; //hard code address for testing
```

```
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                  // network = 0, target = 0 (slave), slave = BROADCAST
  OutMessage [1] = (Address << 5) + 0x01;  //one data bytes
  OutMessage [2] = CAN_WASTE_CON; // message = send system data
  CanOut (OutMessage); // output the message to the CAN
}

void Reset_DO(unsigned char DO_set_point)
{    //clink tells AGUA con to change DO set point
        unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
  OutMessage [1] = (Address << 5) + 0x02;  //two data bytes
  OutMessage [2] = CAN_DO_RESET; // message = send system data
  OutMessage [3] = DO_set_point;
  CanOut (OutMessage); // output the message to the CAN
}

void Reset_Air_rate(unsigned int air_p_set)
{
        unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                  // network = 0, target = 0 (slave), slave = BROADCAST
  OutMessage [1] = (Address << 5) + 0x02;  //two data bytes
  OutMessage [2] = CAN_AIR_FLOW; // message = send system data
  OutMessage [3] = air_p_set;
  CanOut (OutMessage); // output the message to the CAN
}

void UP_PWM0()
{  //updates the PWM0(water pump setting) sent to AGUA-con
        unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                  // network = 0, target = 0 (slave), slave = BROADCAST
  OutMessage [1] = (Address << 5) + 0x02;  //two data bytes
  OutMessage [2] = CAN_PWM0; // message = send PWM0 data
  OutMessage [3] = W_data.Man_PWM0;
  CanOut (OutMessage); // output the message to the CAN
}

void UP_PUMP_CT()
{  // send a msg to AGUA_CON telling it to switch to manual mode for pump
        unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
  OutMessage [1] = (Address << 5) + 0x01;  //one data bytes
  OutMessage [2] = CAN_PUMP_CF; // message = send manual command
  CanOut (OutMessage); // output the message to the CAN
}

void UP_PUMP_CF()
{
```

```
        unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                  // network = 0, target = 0 (slave), slave = BROADCAST
  OutMessage [1] = (Address << 5) + 0x01;  //one data bytes
  OutMessage [2] = CAN_PUMP_CT; // message = send automtic mode command
  CanOut (OutMessage); // output the message to the CAN
}

void Reset_Temp(int new_temp)
{  //sends AGUA con new set point for water temperature
        unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                  // network = 0, target = 0 (slave), slave = BROADCAST
  OutMessage [1] = (Address << 5) + 0x02;  //two data bytes
  OutMessage [2] = CAN_TEMP_SET; // message = change temp set point
  OutMessage [3] = new_temp;
  CanOut (OutMessage); // output the message to the CAN
}

void D_Waste(unsigned int D_Time)
{  //sends message to AGUA con to turn on waste pump for D_Time seconds
        unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                  // network = 0, target = 0 (slave), slave = BROADCAST
  OutMessage [1] = (Address << 5) + 0x02;  //two data bytes
  OutMessage [2] = CAN_DUMP_WASTE; // message = send system data
  OutMessage [3] = D_Time;
  CanOut (OutMessage); // output the message to the CAN

}
void Photo_on(void)
{
        unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                  // network = 0, target = 0 (slave), slave = BROADCAST
  OutMessage [1] = (Address << 5) + 0x01;  //one data byte
  OutMessage [2] = CAN_PHOTO_ON; // message = send system data
  CanOut (OutMessage); // output the message to the CAN

}

void Photo_off(void)
{
        unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                  // network = 0, target = 0 (slave), slave = BROADCAST
  OutMessage [1] = (Address << 5) + 0x01;  //one data byte
  OutMessage [2] = CAN_PHOTO_OFF; // message = send system data
  CanOut (OutMessage); // output the message to the CAN
```

```
}

void Send_DO(void)
{    // sends Clink the DO reading from AGUA feed and water temperature
 unsigned char OutMessage [10];
  OutMessage [0] = 0 * CAN_NETWORK + 1 * CAN_TARGET + (Address >> 3);  //message to clink
  OutMessage [1] = (Address << 5) + 0x03;  //3 data bytes
  OutMessage [2] = CAN_DO_LEVEL; // message = send DO data
  OutMessage [3] = A_data.DO;
  OutMessage [4] = S_data.Water_Temp;
  CanOut (OutMessage); // output the message to the CAN
}

void Send_agua_con_DO(void)
{  //sends the DO reading from Clink to AGUA con
  unsigned char OutMessage [10];
  Address=0x81; //hard code address for testing
  OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (Address >> 3);
                  // network = 0, target = 0 (agua), agua=81 AGUA_con
  OutMessage [1] = (Address << 5) + 0x02;  //2 data bytes
  OutMessage [2] = CAN_HAVE_DO;                  // message = send system data
  OutMessage [3] = A_data.DO;
  CanOut (OutMessage);                    // output the message to the CAN
}


void Feeders(  char *feed_code)    //Serial input from Main PC to activate feeding
{
        unsigned char OutMessage [10];
  char add[9],f_n[9],t_on[9];
  int addr,feeder,time_on;   //addr=address of feeder,feeder=no. of feeder,
                                                        //time_on=.25 sec interval ie. 4=1 sec

        strncpy(add,feed_code,4);
  addr=atoi(add);
  strncpy(f_n,feed_code+4,1);
  feeder=atoi(f_n);
  strncpy(t_on,feed_code+5,3);
  time_on=atoi(t_on);

        OutMessage [0] = 0 * CAN_NETWORK + 0 * CAN_TARGET + (addr >> 3);
                // network = 0, target = 0 (slave), slave = address
        OutMessage [1] = (addr << 5) + 0x03;
                // slave = addresss, message size = 3
        OutMessage [2] = CAN_FEED; // message = feed the culture.
        OutMessage [3] = feeder;
  OutMessage [4] = time_on;
        CanOut (OutMessage); // output the message to the CAN
}

void Send_Error(unsigned char error_code)
{  //sends an error message to CLINK
  unsigned char OutMessage [10];
```

```
  OutMessage [0] = 0 * CAN_NETWORK + 1 * CAN_TARGET + (Address >> 3);  //message to master
        OutMessage [1] = (Address << 5) + 0x01; //message and 1 bytes of data

  switch(error_code)
  {
        case 1: OutMessage [2] = CAN_PUMP_MAL;break; // message = pump malfunction
    case 2: OutMessage [2] = CAN_PWM0_DATA;break; // message = need PWM0 data
    case 3: OutMessage [2] = CAN_SW_MAN;break; // message = switched to manual
  }


        CanOut (OutMessage); // output the message to the CAN unsigned char OutMessage [10];
}

void ask_for_PWM0(void)
{ //sends message from AGUA-con to CLINK to ask for PWM0 from PC
 unsigned char OutMessage [10];

  OutMessage [0] = 0 * CAN_NETWORK + 1 * CAN_TARGET + (Address >> 3);  //message to master
        OutMessage [1] = (Address << 5) + 0x01; //message 1 byte
        OutMessage [2] = CAN_PWM0_DATA; // message = the data of system

        CanOut (OutMessage); // output the message to the CAN

}
//-------------------------End of CAN messages-----------------------------------
//-------------------------------------------------------------------------------
void Feed_Culture (unsigned char feeder,unsigned char time_on)
{
        int sec_count,STOP=1;
  sec_count=QUARTER_SECONDS;
        feed_fin=sec_count+time_on;

  //select feeder,start timer,turn on/off selected feeder
  switch(feeder)
  {
        case 1:
        {while(STOP>=0){Feeder_1=0xFF;STOP=feed_fin-
QUARTER_SECONDS;}Feeder_1=0x0;break;}
    case 2:
        {while(STOP>=0){Feeder_2=0xFF;STOP=feed_fin-
QUARTER_SECONDS;}Feeder_2=0x0;break;}
    case 3:
        {while(STOP>=0){Feeder_3=0;STOP=feed_fin-QUARTER_SECONDS;}Feeder_3=1;break;}
    case 4:
        {while(STOP>=0){Feeder_4=0;STOP=feed_fin-QUARTER_SECONDS;}Feeder_4=1;break;}
  }
}
//--------------------------------control section set up-------------------------
void setup_timer(void)
{
  TM2CON=0x81;  //set up mode to 16-bit overflow with no prescaler ie (14.7456/12)overflows 18.75
times/sec
```

```
 ET2=1;            //enable overflow interupt at 0x73
 T2OV=0;                  //reset the overflow flag
  switch(device)
  {
        case CLINK:time_slice=19;break;// gives 1 sec intervals
     case AGUA_CON:time_slice=19;break;// gives 1 sec intervals
     case AGUA_FEED:time_slice=5;break;//gives .25 sec intervals
  }

}

_interrupt 14 _using 2 void ISR_T2(void)    //timer for system clock
{

        T2OV=0;                     //reset the overflow flag

        if(++T2tick>=time_slice)  //updates every second*(time_slice/19)
  {
          T2tick=0;
        if(device==2)  //AGUA feed timing scheme on the 1/4 sec
        {
     ++QUARTER_SECONDS; //if the device is AGUA feed
     AGUA_FEED_SECONDS=QUARTER_SECONDS/4;  // every 5 ticks = 1/4 sec
     if(++tick_cnt2>=S_data.FD_update_t*4) //updates every FD_update seconds
          {
        FDFLAG=1; //turn on AGUA feed update flag
        tick_cnt2=0;
         }
        }
        else
        ++SECONDS;
                if(++tick_count>=S_data.AC_update_t) //update seconds
                {
                ACFLAG=1;       //turn on AGUA con update flag
        tick_count=0;
        }

  }

}

void setup_pulse_ctn(void)
{
TMOD|=0x05; //sets up T0 as 16 bit counter
TR0=1;  //starts counter0
}

unsigned int get_PULSE(void){
 return(TH0<<8 | TL0); //returns the number of flow count
}

void reset_PULSE(void)
{
TH0=TL0=0;  //resets the count
```

```
}
//----------------------------end of setup-----------------------------------

void Agua_Feed_Update(void)
{
  A_data.DO=get_ADC(07);   //get the data from the sensor and store in memory
  S_data.Water_Temp=GetTemp();
}
//-------------------------------------------------------------------------

void Control_Update(void)
{   //control routine for AGUA con
        int error,dif_press_current;

        PULSE_CNT_METER=get_PULSE();
        PULSE_CNT=60*(PULSE_CNT_METER/S_data.AC_update_t); //pulses/min
        PULSE_LOW=TL0;        //low byte of counter
        PULSE_HIGH=TH0;       //high byte of counter
        reset_PULSE();
   W_data.FR_water_H=PULSE_HIGH;  //update data structure with flow rate
   W_data.FR_water_L=PULSE_LOW;   //update data structure with flow rate
   S_data.Air_Temp=GetTemp();
        if(SECONDS%60==0)Adjust_DO();//adjust DO every min
   Adjust_AIR();
    SerialOutString("ws=  ");
    SerialOutHex (WASTE_C);
    SerialOutString("\n");
   if(WASTE_C==0)      //waste flag is off and regual control
   {
    if(!DIFF_CONTROL)  //if  DIFF_CONTROL=0 and  WASTE_C=0
    {
        dif_press_current=get_ADC(FILTER_DIF);
      SerialOutString("FR=  ");
        SerialOutHex (W_data.FR_Set);
        SerialOutString("\n");
        fil_tot+=get_ADC(FILTER_DIF); //does an A/D conversion for filter diff pressure
        filcount++;
   }
   if(filcount%10==0 && !DIFF_CONTROL)
   {
        F_data.F_Press=fil_tot/10;   //update the ave pressure differential for the filter
        fil_tot=0;
        F_data.F_Press_H=F_data.F_Press>>8;
        F_data.F_Press_L=F_data.F_Press&0xFF;
/*   SerialOutString("filter  ");
   SerialOutHex (F_data.F_Press_H);
   SerialOutHex (F_data.F_Press_L);
   SerialOutString("\n");  */
   }
   else

   if(PULSE_CNT_METER>0 && MAN_PUMP_C==0)MAN_PUMP_C=1;
                                                    //if the flow meter is working and the pump control
                                                    //is set to manual by the program reset to auto
```

```
if(PUMP_C && MAN_PUMP_C)
                // if the automatic mode is set (PUMP_C=1)
                // and not under program "manual" control (MAN_PUMP_C=1)
                // and not under waste management control
               // then control by program "auto" else use manual control
{
//     err_flag_3=0;
                if(PULSE_CNT_METER==0 && F_data.Filter_C_set>0)
                { //if Pulse cnt = 0 (meter stuck?) use diff pressure to control flowrate
      DIFF_CONTROL=1;  //turn on differential flag
     fil_tot+=get_ADC(FILTER_DIF); //does an A/D conversion for filter diff pressure
     filcount++;
     if(filcount%5==0)
     {
      F_data.F_Press=fil_tot/5;    //update the ave pressure differential for the filter
      F_data.F_Press_H=F_data.F_Press>>8;
      F_data.F_Press_L=F_data.F_Press&0xFF;
      fil_tot=0;

                        error=F_data.Filter_C_set-F_data.F_Press;
                        if((error<=6)&&(error>=-6))        //absolute error <= 12
                        {
      //do nothing
                        }
                        else
                        {W_data.Water_Pump=PWM0+=error/4;}//porpotional control
     }
               }
               if(PULSE_CNT_METER)   //if the flow meter is reading
               {
     DIFF_CONTROL=0; //turn off differential flag
                       error=W_data.FR_Set-PULSE_CNT_METER;
                       if((error<=3)&&(error>=-3))        //absolute error <= 8
                       {
                        F_data.Filter_C_set=F_data.F_Press;        //calibrate filter differential set
point
      F_data.Filter_C_set_H=F_data.Filter_C_set>>8;
      F_data.Filter_C_set_L=F_data.Filter_C_set&0xFF;
                       }
     else
                       {W_data.Water_Pump=PWM0+=error/2;}  //porpotional control adjust pump
speed

   }
        if(PWM0>=0xF0)PWM0=0xfe;  //if the pulse rate is near maximum set it to maximun

        if(PULSE_CNT_METER<=0 && F_data.F_Press<=0)  //if both flow meter and diff pressure
read zero
               {
     PWM0=0x0;//turn off pump!
         //take emergency actions!!!!!
    Send_Error(1);//send message by can to clink stating problem
   }
   if(PULSE_CNT_METER==0 && F_data.Filter_C_set<=0)pump_check+=1;
```

```
            //if flow meter is not working
            //and there is no pressure delay for time
      else pump_check=0;
      if(pump_check>=16)
          { //if after 3 checks of system and flow still not working switch to manual
       MAN_PUMP_C=0;
       if(PWM0==0)Send_Error(2);   //send a message to get PWM0 setting from PC
      }
      if(W_data.FR_Set==0)PWM0=0;  //if the flow rate is set to 0 then turn off pump

          //need routine to look for no pressure (ie loss of water)
      Update_data();
    }
    else
//  if(!PUMP_C && err_3_check){err_flag_3=1;err_3_check=0}  //if under manual control
  W_data.Water_Pump=PWM0=W_data.Man_PWM0; //manual control of water pump by program
//    Send_Error(3);
  //send error(3) to tell PC that the device is under manual control
  Update_data();
}

  //do nothing  waste control in effect
}

void Update_data(void)
{



    A_data.Air_Pump=PWM1;
    A_data.A_Press_H=air_ave>>8;
    A_data.A_Press_L=air_ave&0xFF;




}
void Adjust_AIR(void)
{   //
          int i,air_tot=0,air;
          int num_chk=3;
    int error;

          for(i=1;i<=num_chk;i++)
          {

                    air=get_ADC(AIR_PRESS);

                    air_tot+=air;
                    if(i==num_chk)//every num_chk readings calculate the average
                    {
                              air_ave=air_tot/num_chk;
                              air_tot=0;
                    }
```

```
        }
  if(A_data.DO<40) //if the DO meter is extremely low or off line
  {
                error=A_data.A_Set-air_ave;
                if((error<=7)&&(error>=-7))        //absolute error <= 14
                {

                }else
                        {PWM1+=error/3;}        //proportional control change the motor as needed
  }
}

void Adjust_DO(void)
{
        int DO_error,P_test;

        DO_error=(int)A_data.DO_Set-(int)A_data.DO;

  A_data.A_Set=air_ave;

        if((DO_error<=3)&&(DO_error>=-3)) //absolute error <= 6
  {       //do nothing
   A_data.A_Set=air_ave;
  }
   else
  {     //change the PWM of the air pump
   P_test=PWM1+(DO_error/2);  //proportional control change the motor as needed

   if(P_test==0 || P_test>=252)PWM1=3; //if range is close to 0 then 3
   else
   if(P_test>=100)PWM1=100;  //if range > 100 then 100
   else
   if(P_test>3 && P_test<100)PWM1=P_test; //else adjust accordingly
   dummy();
  }
}

void Dump_Waste(int D_Time)
{
        int STOP=1;
  unsigned long sec_count;

        sec_count=SECONDS;
        dump_fin=sec_count+D_Time;

        while(STOP>=0)
                {

                  WASTE_P=0;//turn on waste pump
                  STOP=dump_fin-SECONDS;
                }
        WASTE_P=1;              //turn off waste pump
  WASTE_C=0;    //waste flag is cleared
}
```

```
unsigned int get_ADC(int n_channel)
{
        int nCh;
        nCh=n_channel&0x07;//makes sure channels is between 0-7
        ADCON=0x08|nCh;  //starts ADC
        while(!(ADCON&0x10)); //waits for conversion flag
        ADCON&=0xEF;                    //clears the ADCI
        return((ADCH<<2)|(ADCON>>6)); //reads results and combines to 10 bit
}

int GetTemp(void)
{       //reads the DS1820 digital thermometer and gets temp in C
        int temp;
 TempResetPulse(1);
 TempWriteByte(1,SKIP_ROM);
 TempWriteByte(1,READ_SP);
 temp=TempReadByte(1)/2;
 TempResetPulse(1);
 TempWriteByte(1,SKIP_ROM);
 TempWriteByte(1,CONVERT_T);

 SerialOutString("temp ");
 SerialOutHex(temp);
 SerialOutString("\n");
 return(temp);
}

void check_water_temp(void)
{
        int error;

  error=S_data.Water_Temp_Set-S_data.Water_Temp;

 if((error<1)&&(error>-1))   //deadband +-1
 {
        //do nothing
 }
 else
 if(error<0)
        {
        PWM1=0x00;  //turn off the heater water too warm
    S_data.Heat_Status=0;   //clear status bit
  }
 if(error>0)
  {
        PWM1=0xFF;   //turn on heater water too cold
    S_data.Heat_Status=1;   //clear status bit
  }
 //need more temp sensors may need to dedicate both to system water
 //should check between two sets of tanks in case flow stops to one

}
```

```
void dummy (void)
{

}
//-------------------------------------------------------------------------
//----------------------------CAN routines and such----------------------------
//-------------------------------------------------------------------------
void InitAddress (void)
{
        // Reads and stores the board address
        EA = 0; // disable interrupts
        ADDRESS_SELECT = 0; // enable the reading of the board address
        Address = ADDRESS_PORT; // read the board address from port
        ADDRESS_SELECT = 1; // disable the reading of the board address to free port
        EA = 1; // enable interrupts
}
void InitCan (unsigned char ACode, unsigned int BaudRate)
{
        // initialize the on-chip CAN registers
        // ACode = acceptance code register; BaudRate = baud rate / 100
        // the 11-bit identifier is divided into 3 fields: mm-d-ssssss-sss
        // mm = master; d = direction (0 = m to s; 1 = s to m); s = slave
        // only the 3 most significant bytes of the ACode are relevant; for master
        // number 0, the master should have ACode = 0x20 and the slave ACode = 0x00
        InitAddress (); // Reads the board address
        EA = 0; // disable interrupts
        CANADR = 0; // point to CAN control register (0h)
//      CANDAT = 0x61; // (CR) = no test, 2 sync edges, REF active out, OI disabled
//                                              // EI disabled, TI disabled, RI disabled, reset request
        CANDAT = 0x67; // (CR) = no test, 2 sync edges, REF active out, OI disabled
                                              // EI disabled, TI enabled, RI enabled, reset request
        CANADR = 0x24; // point to acceptance code register (4h) and auto-increment
        CANDAT = ACode; // set acceptance code register
        CANDAT = 0x1F; // set acceptance mask register to 0001 1111
        switch (BaudRate) { // set bit timing depending on selected baud rate

                case 96:
                        #ifdef CLOCK_11_0592
                                CANDAT = 3 * 0x40 + 23; // 11.0592 MHz: SJW=3, BRP=23
                        #endif
                        #ifdef CLOCK_14_7456
                                CANDAT = 3 * 0x40 + 31; // 14.7456 MHz: SJW=3, BRP=31
                        #endif
                        CANDAT = 1 * 0x80 + 6 * 0x10 + 15; // SAM=1, TSEG2=6, TSEG1=15
                break;

                case 128:
                        #ifdef CLOCK_11_0592
                                CANDAT = 3 * 0x40 + 23; // 11.0592 MHz: SJW=3, BRP=23
                        #endif
                        #ifdef CLOCK_14_7456
                                CANDAT = 3 * 0x40 + 31; // 14.7456 MHz: SJW=3, BRP=31
                        #endif
    case 1024:
```

```
                #ifdef CLOCK_11_0592
                        CANDAT = 1 * 0x40 + 2; // 11.0592 MHz: SJW=1, BRP=2
                #endif
                #ifdef CLOCK_14_7456
                        CANDAT = 1 * 0x40 + 3; // 14.7456 MHz: SJW=1, BRP=3
                #endif
                CANDAT = 0 * 0x80 + 4 * 0x10 + 11; // SAM=0, TSEG2=4, TSEG1=11
        break;

        default:
                CANDAT = 0; // SJW=0, BRP=0
                CANDAT = 0; // SAM=0, TSEG2=0, TSEG1=0
    }
    CANDAT = 0x1A; // (OCR) = CTX1=float, CTX0=pu/pl(0=low,1=high), normal output
//CANDAT = 0x1E; // (OCR) = CTX1=float, CTX0=pu/pl(1=low,0=high), normal output
    CANCON = 0xCE; // (CMR) = RX0 active, RX1 active, diff wake-up, no sleep
                                        // clear overrun, release rec buf, abort trans, no trans req
    CANADR = 0; // point to CAN control register (0h)
//  CANDAT = 0x60; // (CR) = no test, 2 sync edges, REF active out, OI disabled
//                                      // EI disabled, TI disabled, RI disabled, start CAN
    CANDAT = 0x66; // (CR) = no test, 2 sync edges, REF active out, OI disabled
                                        // EI disabled, TI enabled, RI enabled, start CAN
    ES1 = 1; // enable CAN interrupt
    EA = 1; // enable interrupts
}

_interrupt (5) _using (1) void CanInterrupt (void)

{
    // interrupt routine to deal with a serial port event
    unsigned char i;
    unsigned char canint = CANCON; // reads the CAN interrupt register
    ERROR_LED = ! ERROR_LED; // blink the CAN communication LED
    if (canint & 0x01) { // receive interrupt
        CANADR = 0x34; // point to start of recive buffer (14h) and auto-increment
        for (i = 0; i < CAN_BUFFER_SIZE; i ++)
            // reads the message to the input buffer
            InBuffer [ActiveInBuf] [i] = CANDAT;
        i = ((InBuffer[ActiveInBuf][0]<<3) + (InBuffer[ActiveInBuf][1]>>5));
            // get the slave address
        if ((InBuffer[ActiveInBuf][0] & CAN_TARGET) || (! i) || (i == Address))
         {
            // if the board is a master or the message is a broadcast or
            // the message destination is the board address, set a flag
            // indicating there is a message ready to be read
            MessageIn [ActiveInBuf] = 1;
            CanMessageIn = 1;
         } else
            MessageIn [ActiveInBuf] = 0;
        if (! MessageIn [! ActiveInBuf]) // if the other message buffer is empty
            ActiveInBuf = ! ActiveInBuf; // switch input buffers
        CANCON = 0xCC; // release receive buffer and clear overrun status
    }
    if (canint & 0x02) { // transmit interrupt
```

```
                if (CanOutFree) // if there is no other message waiting
                        CanOutIdle = 1; // set the idle flag
                else { // if there is another message waiting
                        CANADR = 0x2A; // point to start of transmit buffer (0Ah) and auto-increment
                        for (i = 0; i < CAN_BUFFER_SIZE; i ++)
                                // write the message to the output buffer
                                CANDAT = OutBuffer [i];
                        CANCON = 0xC9; // transmission request and clear overrun status
                        CanOutFree = 1; // signal that the message has been transmitted
                }


        }
}

void FreeCanIn (void) {
        // releases the input buffer with the current message for further input
        EA = 0; // disable interrupts
        MessageIn [! ActiveInBuf] = 0;
                // clearing this flag makes the currently available message invalid
        if (MessageIn [ActiveInBuf])
                // if there is a second message ready to be read, switch input buffers
                // this will make the second message available
                ActiveInBuf = ! ActiveInBuf;
        else
                CanMessageIn = 0;

        EA = 1; // enable interrupts
}

xdat unsigned char * CanIn (void) {
        // returns a pointer to the next available message
        // or 0 if no messages are available
        xdat unsigned char * inbuf;
        EA = 0; // disable interrupts
        inbuf = (MessageIn [! ActiveInBuf] ?
                (xdat unsigned char *) InBuffer [! ActiveInBuf] : 0);
        EA = 1; // enable interrupts
        return (inbuf);
}

void CanOut (unsigned char * message) {
        // transmits a message into the CAN; 'data' points to the start of the
        // message; if there is a message waiting in the queue, it is lost
        unsigned char i;
        EA = 0; // disable interrupts
//      COMM_LED = 0; // turn communication LED on
        if (CanOutIdle) { // if the CAN transmit is idle skip the buffer
                CanOutIdle = 0; // CAN transmit is not idle anymore
                CANADR = 0x2A; // point to start of transmit buffer (0Ah) and auto-increment
                for (i = 0; i < CAN_BUFFER_SIZE; i ++)
                        // writes the message to the output buffer
                        CANDAT = message [i];
                CANCON = 0xC9; // transmission request and clear overrun status
```

```
        } else { // if the CAN transmit is not idle write it to the buffer
                for (i = 0; i < CAN_BUFFER_SIZE; i ++)
                        // writes the message to the CAN output buffer
                        OutBuffer [i] = message [i];
//              CanOutMessage = 1;
                CanOutFree = 0;
        }
        EA = 1; // enable interrupts
}
//-------------------------------------------------------------------------
// Low level routines
//-------------------------------------------------------------------------
// The following routine sends a reset pulse to the temperature sensor ports
//  and detects whether the temperature busses have active sensors attached.
//  If a bus is shorted to Vcc or ground, sensors will not be detected on that
//  bus and the bus will be considered inactive.
// ports = mask specifying to which ports the signal is to be sent (0..0xFF;
//  1 = use port; 0 = do not use port)
// returns: a mask specifying which busses responded to the reset signal and
//  are active and working (0..0xFF; 1 = bus active; 0 = bus inactive)
//-------------------------------------------------------------------------
unsigned char TempResetPulse (unsigned char ports) {
        register unsigned char i;
        register unsigned char presence = 0;
        // Prepare to test the temperature busses
        TEMP_DIR = IN;
        // Make sure all the temperature busses start high:
        #ifdef TEMP_BUS_0
                TEMP_BUS_0 = 1;
        #endif

        // Wait up to MAXWAIT ms until all temperature busses are high:
        for (i = MAXWAIT; i && (!(1
                #ifdef TEMP_BUS_0
                        && TEMP_BUS_0
                #endif

                )); i++) msDelay (1);
        // Test for shorts to ground, disabling bad busses
        //  by masking out the corresponding bit in 'ports':
        #ifdef TEMP_BUS_0
                if (! TEMP_BUS_0) ports &= 0xFE;
        #endif

        // Prepare to send the reset pulse:
        TEMP_DIR = OUT;
        // Send a reset pulse to each active bus:
        #ifdef TEMP_BUS_0
                if (ports & 0x01) TEMP_BUS_0 = 0;
        #endif

        // Delay for at least 480 us:
        usDelay (480);
        // End the reset pulse on each active bus:
```

```
        #ifdef TEMP_BUS_0
                TEMP_BUS_0 = 1;
        #endif

        // Delay for at least 15 us:
        usDelay (15);
        // Prepare to read the presence pulse:
        TEMP_DIR = IN;
        // Try to read the presence pulse on each active bus 10 times:
        for (i = 10; i; i--) {
                #ifdef TEMP_BUS_0
                        if (! TEMP_BUS_0) presence |= 0x01;
                #endif

        }
        // Return the presence pulses detected in the active busses:
        return (presence & ports);
}
//-------------------------------------------------------------------------------
// The following routine writes one bit to the temperature sensor busses.
// ports = mask specifying to which busses the bit is to be sent (0..0xFF)
// wbit = bit to be written (0 or 1)
//-------------------------------------------------------------------------------
void TempWriteBit (unsigned char ports, bit wbit) {
        bit EAstatus;
  register unsigned char i;
        // Prepare to test the temperature busses:
        TEMP_DIR = IN;
        // Make sure all the temperature ports start high:
        #ifdef TEMP_BUS_0
                TEMP_BUS_0 = 1;
        #endif
        // Wait up to MAXWAIT ms until all temperature busses are high:
        for (i = MAXWAIT; i && (!(1
                #ifdef TEMP_BUS_0
                        && TEMP_BUS_0
                #endif

                )); i++) msDelay (1);
        // Test for shorts to ground, disabling bad busses
        //  by masking out the corresponding bit in 'ports':
        #ifdef TEMP_BUS_0
                if (! TEMP_BUS_0) ports &= 0xFE;
        #endif
        // Prepare to write the bit:
        TEMP_DIR = OUT;
        // Store interrupt enable status:
        EAstatus = EA;
        // Disable interrupts:
        EA = 0;
        // For each active bus, pulse the bus low, delay
        //  for more than 1 us and write the bit:
        #ifdef TEMP_BUS_0
                if (ports & 0x01) {
```

```
                        TEMP_BUS_0 = 0;
                        usDelay (1);
                        TEMP_BUS_0 = wbit;
                }
        #endif
        // Restore interrupt enable status:
        EA = EAstatus;
        // Delay for at least 60 us:
        usDelay (60);
        // Prepare to reset the temperature ports:
        TEMP_DIR = IN;
        // Make sure all the temperature ports end high:
        #ifdef TEMP_BUS_0
                TEMP_BUS_0 = 1;
        #endif
}
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// The following routine reads one bit from the temperature sensor busses.
// ports = mask specifying from which busses the bit is to be read (0..0xFF)
// returns: bit read (0 or 1), ANDed between the selected busses
//-----------------------------------------------------------------------------

bit TempReadBit (unsigned char ports) {
        bit EAstatus;
        bit rbit = 1;
  register unsigned char i;
        // Prepare to test the temperature busses:
        TEMP_DIR = IN;
        // Make sure all the temperature ports start high:
        #ifdef TEMP_BUS_0
                TEMP_BUS_0 = 1;
        #endif

        // Wait up to MAXWAIT ms until all temperature busses are high:
        for (i = MAXWAIT; i && (!(1
                #ifdef TEMP_BUS_0
                        && TEMP_BUS_0
                #endif

                )); i++) msDelay (1);
        // Test for shorts to ground, disabling bad busses
        //   by masking out the corresponding bit in 'ports':
        #ifdef TEMP_BUS_0
                if (! TEMP_BUS_0) ports &= 0xFE;
        #endif

        // Read the bit from all active temperature busses:
        #ifdef TEMP_BUS_0
                if (ports & 0x01) {
                        // Prepare to write a pulse:
                        TEMP_DIR = OUT;
                        // Store interrupt enable status:
                        EAstatus = EA;
```

```
                    // Disable interrupts:
                    EA = 0;
                    // Pulse the bus low to signal read:
                    TEMP_BUS_0 = 0;
                    // Delay for at least 1 us:
                    usDelay (1);
                    // Pull the bus back up to prepare for read:
                    TEMP_BUS_0 = 1;
                    // Prepare to read the bit:
                    TEMP_DIR = IN;
                    // Delay for less than 15 us - 7 cycles:
                    usDelay (8);
                    // Read bit and AND it with current result:
                    rbit &= TEMP_BUS_0;
                    // Restore interrupt enable status:
                    EA = EAstatus;
                }
        #endif

        // Return the ANDed result of reading the busses:
        return (rbit);
}

//-----------------------------------------------------------------------------
// Medium level routines
//-----------------------------------------------------------------------------
// The following routine writes one byte to the temperature sensor busses.
// ports = mask specifying to which busses the byte is to be sent (0..0xFF)
// wbyte = byte to be written; the bits are written from low to high
//-----------------------------------------------------------------------------
void TempWriteByte (unsigned char ports, unsigned char wbyte) {
        unsigned char i;
        for (i = 8; i; i--) {
                TempWriteBit (ports, wbyte & 1);
                wbyte >>= 1;
        }
}

//-----------------------------------------------------------------------------
// The following routine reads one byte from the temperature sensor busses.
// ports = mask specifying to which busses the byte is to be sent (0..0xFF)
// returns: byte read, ANDed between busses; bits are read from low to high
//-----------------------------------------------------------------------------
unsigned char TempReadByte (unsigned char ports) {
        unsigned char i;
        unsigned char rbyte = 0;
        for (i = 8; i; i--) {
                rbyte >>= 1;
                rbyte |= TempReadBit (ports) * 0x80;
        }
        return (rbyte);
}
//-----------------------------------------------------------------------------
void Button_Check(void)
```

```
{  /*
                int i;

                if (!(P5 & 0x01))
        {
                  WASTE_P=0;            //turn on waste pump
                  ERROR_LED=0;
        while(!(P5 & 0x01));      //waits for release of button
        for(i=0;i<=100;i++);  //debounce
                  ERROR_LED=1;
        }

                if(!(P5 & 0x02))
        {
                  WASTE_P=1;            //turn  off       waste pump
                  ERROR_LED=0;
        while(!(P5 & 0x02));      //waits for release of button
         for(i=0;i<=100;i++);  //debounce
                  ERROR_LED=1;
        }

                if(!(P5 & 0x04))
        {
                  air_p_set+=5;
                  ERROR_LED=0;
        while(!(P5 & 0x04));      //waits for release of button
                  for(i=0;i<=100;i++); //debounce
                  ERROR_LED=1;
        }
                if(!(P5 & 0x08))
        {
                  air_p_set-=5;
                  ERROR_LED=0;
        while(!(P5 & 0x08));      //waits for release of button
              for(i=0;i<=100;i++); //debounce
                  ERROR_LED=1;
        }
    */
}
```

APPENDIX D
INTERFACE CODE

main.h

```
//---------------------------------------------------------------------------
#ifndef mainH
#define mainH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
//---------------------------------------------------------------------------
class TMain_Form : public TForm
{
__published:      // IDE-managed Components
        TButton *Photo;
        TButton *filter;
        TButton *Button3;
        TButton *oxy;
        TButton *growth;
        TButton *Button6;
        TButton *Button7;
        TButton *Button8;
        TBitBtn *BitBtn1;
        TButton *Button1;
        TButton *Waste_Treat;
        TLabel *Label1;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall PhotoClick(TObject *Sender);
        void __fastcall filterClick(TObject *Sender);
        void __fastcall Button3Click(TObject *Sender);
        void __fastcall oxyClick(TObject *Sender);
        void __fastcall growthClick(TObject *Sender);
        void __fastcall Button6Click(TObject *Sender);
        void __fastcall Button7Click(TObject *Sender);
        void __fastcall Button8Click(TObject *Sender);
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
```

```
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall Waste_TreatClick(TObject *Sender);


private:  // User declarations
public:           // User declarations

        __fastcall TMain_Form(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TMain_Form *Main_Form;
//---------------------------------------------------------------------------
#endif
```

system_status.h

```
//---------------------------------------------------------------------------
#ifndef system_statusH
#define system_statusH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\ExtCtrls.hpp>
#include <vcl\Buttons.hpp>
#include "comport.h"
#include <vcl\ComCtrls.hpp>
//---------------------------------------------------------------------------
class Tsys_status : public TForm
{
__published:    // IDE-managed Components
        TTimer *Get_Data;
        TSpeedButton *Light_Status;
        TStatusBar *StatusBar;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TEdit *sys_flow;
        TEdit *sys_set;
        TEdit *pwm_level;
        TEdit *air_P;
        TEdit *filter_P;
        TLabel *Lights;
        TComPort *ComPort;
        TButton *Sys_Set_FR;
        TLabel *Label6;
        TLabel *Label7;
```

```
TEdit *air_set;
TButton *Air_Change;
TLabel *monl;
TLabel *Label8;
TLabel *dayl;
TLabel *Label10;
TLabel *yearl;
TLabel *hourl;
TLabel *Label13;
TLabel *minl;
TLabel *Label15;
TLabel *secl;
TTimer *Waste_Timer;
TBitBtn *Dump_W;
TEdit *Dump_Time;
TLabel *Label9;
TEdit *A_P_L;
TTimer *Timer2;
TButton *Visual;
TEdit *sys_reset;
TEdit *Air_Reset;
TTimer *Timer3;
TTimer *Timer4;
TLabel *Label12;
TLabel *Label14;
TEdit *O2_level;
TButton *reset_DO;
TLabel *Label16;
TEdit *set_DO;
TEdit *DO_reset;
TLabel *Label18;
TEdit *fil_set_P;
TLabel *Label19;
TEdit *Water_Temp;
TLabel *Label20;
TEdit *W_T_Set;
TLabel *Label21;
TEdit *A_temp;
TButton *Temp_Reset;
TEdit *T_Reset;
TLabel *Label22;
TSpeedButton *Heat_Status;
TTimer *Feed_Interval;
TLabel *Label11;
TLabel *Label17;
TButton *P_Man;
TEdit *P_Man_V;
TButton *PUMP_CONTROL;
TLabel *Label23;
TTimer *Screen_Update;
TTimer *M_P_D;
TTimer *M_W_T;
TTimer *R_P;
void __fastcall Get_DataTimer(TObject *Sender);
```

```
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall ComPortWriteByte(TObject *Sender, char &ReceivedByte);
        void __fastcall ComPortReadLine(TObject *Sender, char *ReceivedLine);
        void __fastcall Light_StatusClick(TObject *Sender);
        void __fastcall send_PWM0_data( int PWM0);


        void __fastcall Sys_Set_FRClick(TObject *Sender);
        void __fastcall Air_ChangeClick(TObject *Sender);
        void __fastcall Waste_TimerTimer(TObject *Sender);
        void __fastcall Dump_WClick(TObject *Sender);
        void __fastcall Timer2Timer(TObject *Sender);
        void __fastcall VisualClick(TObject *Sender);
        void __fastcall Timer3Timer(TObject *Sender);
        void __fastcall Timer4Timer(TObject *Sender);

        void __fastcall reset_DOClick(TObject *Sender);
        void __fastcall Temp_ResetClick(TObject *Sender);
        void __fastcall Feed_IntervalTimer(TObject *Sender);


        void __fastcall P_ManClick(TObject *Sender);
        void __fastcall PUMP_CONTROLClick(TObject *Sender);


        void __fastcall FormActivate(TObject *Sender);
        void __fastcall Screen_UpdateTimer(TObject *Sender);
        void __fastcall M_P_DTimer(TObject *Sender);
        void __fastcall M_W_TTimer(TObject *Sender);
        void __fastcall R_PTimer(TObject *Sender);
private: // User declarations
public:         // User declarations
        __fastcall Tsys_status(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern Tsys_status *sys_status;

//---------------------------------------------------------------------------
#endif
```

growth.h

```
//---------------------------------------------------------------------------
#ifndef growthH
#define growthH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\ChartFX.hpp>
#include <vcl\OleCtrls.hpp>
#include <vcl\Mask.hpp>
```

```
#include <vcl\Buttons.hpp>
#include <vcl\Tabnotbk.hpp>
#include <vcl\ComCtrls.hpp>

//---------------------------------------------------------------------------
class T_growth : public TForm
{
__published:    // IDE-managed Components
        TTabbedNotebook *Tanks;
        TLabel *T4_sp_name;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TSpeedButton *SpeedButton1;
        TLabel *Label7;
        TEdit *T4_wt;
        TEdit *T4_I_T;
        TEdit *T4_grw_f;
        TEdit *T4_b_fac;
        TButton *Button1;
        TChartFX *T4_ChartFX;
        TEdit *T4_D_O_C;
        TEdit *T4_W_C;
        TLabel *Label8;
        TEdit *T4_S_W;
        TLabel *T3_sp_name;
        TLabel *Label10;
        TLabel *Label11;
        TLabel *Label12;
        TLabel *Label13;
        TLabel *Label14;
        TLabel *Label15;
        TSpeedButton *SpeedButton2;
        TLabel *Label16;
        TEdit *T3_grw_f;
        TEdit *T3_b_fac;
        TButton *Button2;
        TChartFX *T3_ChartFX;
        TEdit *T3_D_O_C;
        TEdit *T3_W_C;
        TLabel *Label17;
        TEdit *T3_S_W;
        TLabel *T2_sp_name;
        TLabel *Label19;
        TLabel *Label20;
        TLabel *Label21;
        TLabel *Label22;
        TLabel *Label23;
        TLabel *Label24;
        TSpeedButton *SpeedButton3;
        TEdit *T2_wt;
```

```
TEdit *T2_I_T;
TEdit *T2_grw_f;
TEdit *T2_b_fac;
TButton *Button3;
TChartFX *T2_ChartFX;
TEdit *T2_D_O_C;
TEdit *T2_W_C;
TLabel *Label26;
TLabel *T1_sp_N;
TLabel *Label28;
TLabel *Label29;
TLabel *Label30;
TLabel *Label31;
TLabel *Label32;
TLabel *Label33;
TSpeedButton *SpeedButton4;
TEdit *T1_wt;
TEdit *T1_I_T;
TEdit *T1_grw_f;
TEdit *T1_b_fac;
TButton *Button4;
TChartFX *T1_ChartFX;
TEdit *T1_D_O_C;
TEdit *T1_W_C;
TLabel *Label35;
TEdit *T1_S_W;
TEdit *T4_S_D;
TEdit *T3_S_D;
TEdit *T2_S_D;
TEdit *T1_S_D;
TButton *T4_Store;
TButton *T3_Store;
TButton *T2_Store;
TButton *T1_Store;
TLabel *Label9;
TEdit *T2_S_W;
TEdit *Edit1;
TEdit *Edit2;
TEdit *Edit3;
TEdit *Edit4;
TEdit *Edit5;
TEdit *T3_I_T;
TEdit *T3_wt;
void __fastcall Button1Click(TObject *Sender);
void __fastcall T4_I_TChange(TObject *Sender);

void __fastcall SpeedButton1Click(TObject *Sender);


void __fastcall T4_S_DChange(TObject *Sender);

void __fastcall T2_I_TChange(TObject *Sender);
void __fastcall T3_I_TChange(TObject *Sender);
void __fastcall T1_I_TChange(TObject *Sender);
```

```
        void __fastcall T1_wtChange(TObject *Sender);
        void __fastcall T1_S_DChange(TObject *Sender);
        void __fastcall T1_grw_fChange(TObject *Sender);
        void __fastcall T1_b_facChange(TObject *Sender);
        void __fastcall T1_S_WChange(TObject *Sender);

        void __fastcall T1_D_O_CChange(TObject *Sender);
        void __fastcall T4_b_facChange(TObject *Sender);
        void __fastcall T4_grw_fChange(TObject *Sender);
        void __fastcall T3_grw_fChange(TObject *Sender);
        void __fastcall T3_b_facChange(TObject *Sender);
        void __fastcall T2_grw_fChange(TObject *Sender);
        void __fastcall T2_b_facChange(TObject *Sender);
        void __fastcall T2_StoreClick(TObject *Sender);
        void __fastcall T2_D_O_CChange(TObject *Sender);
        void __fastcall T3_D_O_CChange(TObject *Sender);
        void __fastcall T4_D_O_CChange(TObject *Sender);
        void __fastcall T1_StoreClick(TObject *Sender);
        void __fastcall T3_StoreClick(TObject *Sender);
        void __fastcall T4_StoreClick(TObject *Sender);


        void __fastcall TanksEnter(TObject *Sender);

        void __fastcall T2_wtChange(TObject *Sender);

        void __fastcall T2_S_DChange(TObject *Sender);
        void __fastcall T2_S_WChange(TObject *Sender);
        void __fastcall T3_wtChange(TObject *Sender);
        void __fastcall T3_S_DChange(TObject *Sender);
        void __fastcall T3_S_WChange(TObject *Sender);

        void __fastcall T4_wtChange(TObject *Sender);
        void __fastcall T4_S_WChange(TObject *Sender);


private:  // User declarations
public:         // User declarations
        __fastcall T_growth(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern T_growth *_growth;
//---------------------------------------------------------------------------
#endif
```

_feed.h

```
//---------------------------------------------------------------------------
#ifndef _feedH
#define _feedH
```

```
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\ExtCtrls.hpp>
#include <vcl\ComCtrls.hpp>
#include <vcl\Mask.hpp>
#include <vcl\Buttons.hpp>
//---------------------------------------------------------------------------
class TFeed_sch : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TEdit *AF_add;
        TPageControl *Periodic;
        TTabSheet *TabSheet1;
        TTabSheet *TabSheet2;
        TTabSheet *TabSheet3;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TMaskEdit *MaskEdit1;
        TMaskEdit *MaskEdit2;
        TLabel *Label5;
        TEdit *Edit2;
        TLabel *Label6;
        TEdit *Edit3;
        TLabel *Label7;
        TEdit *Edit4;
        TMaskEdit *MaskEdit3;
        TMaskEdit *MaskEdit4;
        TLabel *Label8;
        TLabel *Label9;
        TEdit *Edit5;
        TLabel *Label10;
        TEdit *Edit6;

        TTabSheet *calibrate;
        TLabel *Label11;
        TSpeedButton *cal_feeder;
        TEdit *Cal_time;
        TLabel *Label12;
        TLabel *Label13;
        TGroupBox *Feeder1;
        TLabel *Label14;
        TGroupBox *Feeder3;
        TGroupBox *Feeder4;
        TLabel *Label16;
        TLabel *Label17;
    TEdit *M_F_T1;
        TEdit *M_F_T3;
        TEdit *M_F_T4;
        TLabel *Label18;
```

```
TLabel *Label20;
TLabel *Label21;
TEdit *F_I1;
TEdit *F_I3;
TEdit *F_I4;
TSpeedButton *P_F1;
TSpeedButton *P_F3;
TSpeedButton *P_F4;
TEdit *PT1;
TEdit *PT3;
TEdit *PT4;
TTimer *Timer1;
TLabel *Label22;
TLabel *Label24;
TLabel *Label25;
TLabel *Label26;
TLabel *Label27;
TLabel *Label28;
TSpeedButton *SpeedButton2;

TGroupBox *Feeder2;
TLabel *Label15;
TLabel *Label19;
TSpeedButton *P_F2;
TLabel *Label23;
TEdit *M_F_T2;
TEdit *F_I2;
TEdit *PT2;
TSpeedButton *Log_Sch;
TEdit *AMT_D;
TLabel *Label29;
TEdit *feed_num;
TUpDown *FN_Change;
TLabel *Label30;
TLabel *Label31;
TButton *Cal_Feed;
TButton *Set_Calibration;
TLabel *Label32;
TLabel *FN;
TLabel *Label33;
TLabel *CAL_SET;
TLabel *Label34;
TTimer *Timer2;
TEdit *Edit7;
TEdit *Edit8;
TEdit *Edit9;
TLabel *Label35;
TEdit *T1BW;
TLabel *Label36;
TEdit *T2BW;
TLabel *Label37;
TEdit *T3BW;
TLabel *Label38;
TEdit *T4BW;
```

```
TButton *Button1;
TEdit *Edit1;
TEdit *Edit10;
TLabel *Label39;
TEdit *T1_FW;
TLabel *Label40;
TEdit *T1_NF;
TLabel *Label41;
TEdit *T1_AF;
TEdit *T2_FW;
TLabel *Label42;
TLabel *Label43;
TEdit *T2_NF;
TLabel *Label44;
TEdit *T2_AF;
TEdit *T3_FW;
TLabel *Label45;
TLabel *Label46;
TEdit *T3_NF;
TLabel *Label47;
TEdit *T3_AF;
TEdit *T4_FW;
TLabel *Label48;
TLabel *Label49;
TEdit *T4_NF;
TLabel *Label50;
TEdit *T4_AF;
TEdit *cal_1;
TEdit *cal_2;
TEdit *cal_3;
TEdit *cal_4;
TLabel *Label51;
TLabel *Label52;
TLabel *Label53;
TLabel *Label54;
TLabel *Label55;
TLabel *Label56;
TLabel *Label57;
TLabel *Label58;
TSpeedButton *A_Mode1;
TSpeedButton *A_Mode2;
TSpeedButton *A_Mode3;
TSpeedButton *A_Mode4;
void __fastcall AF_addChange(TObject *Sender);
void __fastcall P_F1MouseDown(TObject *Sender, TMouseButton Button,
TShiftState Shift, int X, int Y);
void __fastcall P_F1MouseUp(TObject *Sender, TMouseButton Button,
TShiftState Shift, int X, int Y);
void __fastcall P_F2MouseDown(TObject *Sender, TMouseButton Button,
TShiftState Shift, int X, int Y);
void __fastcall P_F2MouseUp(TObject *Sender, TMouseButton Button,
TShiftState Shift, int X, int Y);
void __fastcall P_F3MouseDown(TObject *Sender, TMouseButton Button,
TShiftState Shift, int X, int Y);
```

```
        void __fastcall P_F3MouseUp(TObject *Sender, TMouseButton Button,
        TShiftState Shift, int X, int Y);
        void __fastcall P_F4MouseDown(TObject *Sender, TMouseButton Button,
        TShiftState Shift, int X, int Y);
        void __fastcall P_F4MouseUp(TObject *Sender, TMouseButton Button,
        TShiftState Shift, int X, int Y);

        void __fastcall M_F_T1Change(TObject *Sender);
void __fastcall M_F_T2Change(TObject *Sender);
        void __fastcall M_F_T3Change(TObject *Sender);
        void __fastcall M_F_T4Change(TObject *Sender);
        void __fastcall F_l1Change(TObject *Sender);
        void __fastcall F_l2Change(TObject *Sender);
        void __fastcall F_l3Change(TObject *Sender);
        void __fastcall F_l4Change(TObject *Sender);
        void __fastcall PT1Change(TObject *Sender);
        void __fastcall PT2Change(TObject *Sender);
        void __fastcall PT3Change(TObject *Sender);
        void __fastcall PT4Change(TObject *Sender);
        void __fastcall Timer1Timer(TObject *Sender);
        void __fastcall M_F_T1Exit(TObject *Sender);
        void __fastcall M_F_T2Exit(TObject *Sender);
        void __fastcall M_F_T3Exit(TObject *Sender);
        void __fastcall M_F_T4Exit(TObject *Sender);



        void __fastcall Log_SchClick(TObject *Sender);

        void __fastcall FormActivate(TObject *Sender);

        void __fastcall FN_ChangeClick(TObject *Sender, TUDBtnType Button);
        void __fastcall cal_feederMouseDown(TObject *Sender, TMouseButton Button,
        TShiftState Shift, int X, int Y);

        void __fastcall cal_feederMouseUp(TObject *Sender, TMouseButton Button,
        TShiftState Shift, int X, int Y);



        void __fastcall Cal_FeedClick(TObject *Sender);
        void __fastcall Cal_timeExit(TObject *Sender);
        void __fastcall Set_CalibrationClick(TObject *Sender);
        void __fastcall Timer2Timer(TObject *Sender);
        void __fastcall AMT_DExit(TObject *Sender);
        void __fastcall SpeedButton2Click(TObject *Sender);



        void __fastcall T1BWChange(TObject *Sender);
        void __fastcall T2BWChange(TObject *Sender);
        void __fastcall T3BWChange(TObject *Sender);
        void __fastcall T4BWChange(TObject *Sender);
        void __fastcall Button1Click(TObject *Sender);
```

```
        void __fastcall T1_NFChange(TObject *Sender);
        void __fastcall T2_NFChange(TObject *Sender);
        void __fastcall T3_NFChange(TObject *Sender);
        void __fastcall T4_NFChange(TObject *Sender);




        void __fastcall TabSheet3Enter(TObject *Sender);

        void __fastcall T3_FWChange(TObject *Sender);
        void __fastcall T1_FWChange(TObject *Sender);
        void __fastcall T2_FWChange(TObject *Sender);
        void __fastcall T4_FWChange(TObject *Sender);


        void __fastcall A_Mode1Click(TObject *Sender);
        void __fastcall A_Mode2Click(TObject *Sender);
        void __fastcall A_Mode3Click(TObject *Sender);
        void __fastcall A_Mode4Click(TObject *Sender);
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
private: // User declarations
public:         // User declarations


        __fastcall TFeed_sch(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TFeed_sch *Feed_sch;
//---------------------------------------------------------------------------
#endif
```

_oxygen.h

```
//---------------------------------------------------------------------------
#ifndef _oxygenH
#define _oxygenH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
//---------------------------------------------------------------------------
class ToxyF : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TEdit *sp_min;
        TEdit *sp_max;
        TLabel *Label3;
```

```
        TEdit *max_level;
        TLabel *Label4;
        TEdit *temp;
        TLabel *Label5;
        TEdit *sal;
        TLabel *Label6;
        TEdit *el_M;
        TButton *calc_oxy;
        TLabel *Label7;
        TEdit *water_O2;
        TLabel *_spN;
        void __fastcall calc_oxyClick(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
private: // User declarations
public:         // User declarations
        __fastcall ToxyF(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern ToxyF *oxyF;
//---------------------------------------------------------------------------
#endif
```

_system_design.h

```
//---------------------------------------------------------------------------
#ifndef _system_designH
#define _system_designH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\ExtCtrls.hpp>
//---------------------------------------------------------------------------
class Tsystem_design : public TForm
{
__published:    // IDE-managed Components
        TRadioGroup *Tank_Type;
        TEdit *tank_vol;
        TLabel *Label1;
        TLabel *Label2;
        TEdit *n_t;
        TLabel *Label3;
        TEdit *tank_ht;
        TLabel *Label4;
        TEdit *tank_wd;
        TLabel *t_l;
        TEdit *tank_len;
        TRadioGroup *Units;
        TLabel *Label5;
        void __fastcall Tank_TypeClick(TObject *Sender);
        void __fastcall tank_htChange(TObject *Sender);
        void __fastcall tank_wdChange(TObject *Sender);
```

```
        void __fastcall tank_lenChange(TObject *Sender);
        void __fastcall n_tChange(TObject *Sender);


        void __fastcall UnitsClick(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);

private:  // User declarations
public:         // User declarations
        __fastcall Tsystem_design(TComponent* Owner);
   void __fastcall Tsystem_design::calc_vol(void);
};
//---------------------------------------------------------------------------
extern Tsystem_design *system_design;

//---------------------------------------------------------------------------
#endif
```

_temperature.h

```
//---------------------------------------------------------------------------
#ifndef _temperatureH
#define _temperatureH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
//---------------------------------------------------------------------------
class Ttemperature : public TForm
{
__published:    // IDE-managed Components
private:  // User declarations
public:         // User declarations
        __fastcall Ttemperature(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern Ttemperature *temperature;
//---------------------------------------------------------------------------
#endif
```

err_F.h

```
//---------------------------------------------------------------------------
#ifndef err_FH
#define err_FH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
//---------------------------------------------------------------------------
class TError_F : public TForm
```

```
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:  // User declarations
public:         // User declarations
        __fastcall TError_F(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TError_F *Error_F;
//---------------------------------------------------------------------------
#endif
```

filter_design.h

```
//---------------------------------------------------------------------------
#ifndef filter_designH
#define filter_designH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\ExtCtrls.hpp>
#include <vcl\Buttons.hpp>
#include <math.h>

//---------------------------------------------------------------------------
class Tfilter_form : public TForm
{
__published:    // IDE-managed Components
        TRadioGroup *Part_size;
        TEdit *Show_p_size;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TEdit *filter_h;
        TEdit *Filter_w;
        TLabel *Label4;
        TEdit *weight_of_media;
        TBitBtn *calc;
        TEdit *S_G;
        TLabel *Label5;
        TLabel *Label6;
        TEdit *TSA;
        TRadioGroup *Media;
        TLabel *Label7;
        TEdit *media_type;
        TEdit *V_R;
        TLabel *Label8;
        TLabel *Label9;
        TEdit *MV;
```

```
        void __fastcall Part_sizeClick(TObject *Sender);
        void __fastcall Show_p_sizeChange(TObject *Sender);
        void __fastcall calcClick(TObject *Sender);
        void __fastcall weight_of_mediaChange(TObject *Sender);

        void __fastcall MediaClick(TObject *Sender);
        void __fastcall V_RChange(TObject *Sender);
private:  // User declarations
public:         // User declarations
        __fastcall Tfilter_form(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern Tfilter_form *filter_form;
//---------------------------------------------------------------------------
#endif
```

Graph_Output.h

```
//---------------------------------------------------------------------------
#ifndef Graph_OutputH
#define Graph_OutputH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\ChartFX.hpp>
#include <vcl\OleCtrls.hpp>
#include <vcl\ExtCtrls.hpp>
#include <vcl\ComCtrls.hpp>
//---------------------------------------------------------------------------
class TVisual_Output : public TForm
{
__published:    // IDE-managed Components
        TChartFX *W_Pump;
        TTimer *Scan_Pump;
        TProgressBar *Water_Pump_PWM;
        TLabel *Label11;
        TLabel *Label2;
        TLabel *WP_PWM;
        TChartFX *Air_P;
        TProgressBar *Air_Pump_PWM;
        TLabel *Label4;
        TLabel *AP_PWM;
        TChartFX *Diff_Press;
        TChartFX *Temp_vs_Set;
        TLabel *Label3;
        TLabel *Label5;
        void __fastcall Scan_PumpTimer(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall CheckBox2Click(TObject *Sender);
        void __fastcall CheckBox1Click(TObject *Sender);
```

```
        void __fastcall CheckBox3Click(TObject *Sender);


        void __fastcall CheckBox6Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
private:  // User declarations
 int nPoints.nStep,nCounter.nScrollLegend;
public:                 // User declarations
        __fastcall TVisual_Output(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TVisual_Output *Visual_Output;
//---------------------------------------------------------------------------
#endif
```

photo.h

```
//---------------------------------------------------------------------------
#ifndef photoH
#define photoH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\ComCtrls.hpp>
#include <vcl\ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TphotoF : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TEdit *location;
        TEdit *latitude;
        TEdit *longitude;
        TLabel *sun_pos;
        TLabel *Label9;
        TLabel *Label10;
        TLabel *Label11;
        TTimer *Timer1;
        TLabel *time_hr;
        TLabel *Label6;
        TLabel *time_min;
        TLabel *date_month;
        TLabel *Label15;
        TLabel *date_day;
        TLabel *Label17;
        TLabel *date_year;
```

```
        TLabel *Label16;
        TLabel *day_y;
        TEdit *_sunrise;
        TEdit *_sunset;
        TEdit *_day_length;
        TLabel *Label_day;
        void __fastcall Timer1Timer(TObject *Sender);




private:  // User declarations
public:         // User declarations
        __fastcall TphotoF(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TphotoF *photoF;
//---------------------------------------------------------------------------
#endif
```

pump_error.h

```
//---------------------------------------------------------------------------
#ifndef pump_errorH
#define pump_errorH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TPUMP_ERR : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TBitBtn *BitBtn1;
        TTimer *pump_err_timer;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall pump_err_timerTimer(TObject *Sender);
private:  // User declarations
public:         // User declarations
        __fastcall TPUMP_ERR(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TPUMP_ERR *PUMP_ERR;
//---------------------------------------------------------------------------
#endif
```

species.h

```
//---------------------------------------------------------------------------
#ifndef speciesH
#define speciesH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\ExtCtrls.hpp>
#include <vcl\ComCtrls.hpp>
//---------------------------------------------------------------------------
class Tselect_species : public TForm
{
__published:    // IDE-managed Components
        TBitBtn *close;
        TRadioGroup *species_parameters;
        TLabel *Label1;
        TEdit *T_N;
        TUpDown *T_N_C;
        TLabel *Label2;
        void __fastcall closeClick(TObject *Sender);

        void __fastcall species_parametersClick(TObject *Sender);
        void __fastcall T_N_CClick(TObject *Sender, TUDBtnType Button);


        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall FormActivate(TObject *Sender);
private:  // User declarations
public:         // User declarations
        __fastcall Tselect_species(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern Tselect_species *select_species;
//---------------------------------------------------------------------------
#endif
```

waste.h

```
//---------------------------------------------------------------------------
#ifndef WasteH
#define WasteH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\ComCtrls.hpp>
#include <vcl\ExtCtrls.hpp>
//---------------------------------------------------------------------------
```

```
class TW_M : public TForm
{
__published:      // IDE-managed Components
        TLabel *Label1;
        TEdit *PFR;
        TLabel *Label2;
        TEdit *DER;
        TLabel *Label3;
        TEdit *IBE;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TEdit *NO3L;
        TLabel *Label8;
        TEdit *NO2L;
        TLabel *Label9;
        TLabel *Label10;
        TLabel *Label11;
        TLabel *Label12;
        TEdit *Edit1;
        TEdit *Edit2;
        TLabel *Label13;
        TLabel *Label14;
        TStatusBar *Recomendations;
        TTimer *Timer1;
        TButton *Start;
        TLabel *Label15;
        TLabel *Label16;
        TLabel *Label17;
        TLabel *Label18;
        TEdit *T_P_O;
        TTimer *Timer2;
        TButton *File_Save;
        TLabel *Label19;
        TLabel *Label20;
        TLabel *Label21;
        void __fastcall FormActivate(TObject *Sender);

        void __fastcall PFRChange(TObject *Sender);
        void __fastcall DERChange(TObject *Sender);
        void __fastcall IBEChange(TObject *Sender);
        void __fastcall NO3LChange(TObject *Sender);
        void __fastcall NO2LChange(TObject *Sender);
        void __fastcall Timer1Timer(TObject *Sender);
        void __fastcall StartClick(TObject *Sender);
        void __fastcall Timer2Timer(TObject *Sender);
        void __fastcall File_SaveClick(TObject *Sender);

private:  // User declarations
public:           // User declarations
        __fastcall TW_M(TComponent* Owner);
};
//---------------------------------------------------------------------------
```

```
extern TW_M *W_M;

//---------------------------------------------------------------------------
#endif
```

Comport.h

```
//---------------------------------------------------------------------------
#ifndef ComPortH
#define ComPortH
//---------------------------------------------------------------------------
#include <vcl\SysUtils.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Forms.hpp>
//---------------------------------------------------------------------------
#define DEFAULT_BUFFER_SIZE 256
class TReadComPortLoop;
class TWriteComPortLoop;
//---------------------------------------------------------------------------
class TComPort : public TComponent
{
        friend TReadComPortLoop;
        friend TWriteComPortLoop;
private:
        enum TPortNumber { NUL, COM1, COM2, COM3, COM4 };
                static const char * PortName [5];
                TPortNumber FPort;
        DWORD FBaudRate;
                void __fastcall SetBaudRate (DWORD value);
        enum TByteSize { _5_Bits, _6_Bits, _7_Bits, _8_Bits };
                TByteSize FByteSize;
                void __fastcall SetByteSize (TByteSize value);
        enum TParity { NoParity, OddParity, EvenParity, MarkParity, SpaceParity };
                TParity FParity;
        enum TStopBits { _1_Stop, _1_5_Stop, _2_Stop };
                TStopBits FStopBits;
                void __fastcall SetStopBits (TStopBits value);
        DWORD FBufferSize;
        char FEndOfLineCharacter;
        typedef void __fastcall (__closure * TOnReadByte) (TObject *Sender, char & ReceivedByte);
                TOnReadByte FOnReadByte;
        typedef void __fastcall (__closure * TOnReadLine) (TObject *Sender, char * ReceivedLine);
                TOnReadLine FOnReadLine;
        typedef void __fastcall (__closure * TOnWriteByte) (TObject *Sender, char & ByteToSend);
                TOnWriteByte FOnWriteByte;
        HANDLE hComPort; // file handle of the comm port
        HANDLE hWriteComPortDone; // event object signaled when a write to the comm port is
complete
        HANDLE hReadComPortDone; // event object signaled when a character was read from the
comm port
```

```
        HANDLE hCharactersToWrite; // event object signaled when a character is put in the output
buffer
        OVERLAPPED ComPortWriteOL; // asynchronous IO information for writing to the comm port
        OVERLAPPED ComPortReadOL; // asynchronous IO information for reading from the comm
port
        TReadComPortLoop * ReadComPortLoop;
        TWriteComPortLoop * WriteComPortLoop;
        DWORD OpenBufferSize;
        char * InputBuffer;
                DWORD BytesRead;
        char * OutputBuffer;
                DWORD NextByteOut;
                DWORD LastByteOut;
protected:
        void __fastcall ReadByte (void);
        void __fastcall ReadLine (void);
        void __fastcall WriteByte (void);
public:
        __fastcall TComPort(TComponent* Owner);
        __fastcall ~TComPort();
        void __fastcall Open (void);
        void __fastcall Close (void);
        void __fastcall Write (System::AnsiString Line);
__published:
        __property TPortNumber Port = {read=FPort, write=FPort};
        __property DWORD BaudRate = {read=FBaudRate, write=SetBaudRate, default=9600};
        __property TByteSize ByteSize = {read=FByteSize, write=SetByteSize, default=_8_Bits};
        __property TParity Parity = {read=FParity, write=FParity, default=NoParity};
        __property TStopBits StopBits = {read=FStopBits, write=SetStopBits, default=_1_Stop};
        __property DWORD BufferSize = {read=FBufferSize, write=FBufferSize,
default=DEFAULT_BUFFER_SIZE};
        __property char EndOfLineCharacter = {read=FEndOfLineCharacter,
write=FEndOfLineCharacter, default='\n'};
        __property TOnReadByte OnReadByte = {read=FOnReadByte, write=FOnReadByte};
        __property TOnReadLine OnReadLine = {read=FOnReadLine, write=FOnReadLine};
        __property TOnWriteByte OnWriteByte = {read=FOnWriteByte, write=FOnWriteByte};
};
//---------------------------------------------------------------------------
class TReadComPortLoop : public TThread {
private:
        TComPort * ComPort;
protected:
        void __fastcall Execute();
public:
        __fastcall TReadComPortLoop(TComPort * Port);
};
//---------------------------------------------------------------------------
class TWriteComPortLoop : public TThread {
private:
        TComPort * ComPort;
protected:
        void __fastcall Execute();
public:
        __fastcall TWriteComPortLoop(TComPort * Port);
```

```
};
//---------------------------------------------------------------------------
#endif
```

_carp.h

```
//---------------------------------------------------------------------------
#ifndef _carpH
#define _carpH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\Mask.hpp>
//---------------------------------------------------------------------------
class TCarp : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TEdit *low_temp;
        TEdit *low_DO;
        TEdit *low_pH;
        TEdit *low_alk;
        TEdit *low_CO2;
        TEdit *low_NH4;
        TEdit *low_NO2;
        TEdit *low_hardness;
        TEdit *low_chloride;
        TEdit *low_salinity;
        TBitBtn *BitBtn1;
        TEdit *high_temp;
        TEdit *high_DO;
        TEdit *high_pH;
        TEdit *high_alk;
        TEdit *high_CO2;
        TEdit *high_NH4;
        TEdit *high_NO2;
        TEdit *high_hardness;
        TEdit *high_chloride;
        TEdit *high_salinity;
        TLabel *Label11;
```

```
        TLabel *Label12;
        TLabel *Label13;
        TLabel *Label15;
        TLabel *Label16;
        TLabel *Label17;
        TLabel *Label18;
        TLabel *Label19;
        TLabel *Label20;
        TMaskEdit *growth_factor;
        TEdit *b_factor;
        TEdit *weight_fin;
        TEdit *S_Wt;
        TEdit *S_day;
        TEdit *D_U_H;
        TLabel * _spN;
        TLabel *Label14;
        TEdit *D_O_C;
        TLabel *Label21;
        TLabel *Label22;
        TLabel *Label23;
        TEdit *M_W;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall S_WtChange(TObject *Sender);
        void __fastcall growth_factorChange(TObject *Sender);


        void __fastcall weight_finChange(TObject *Sender);
        void __fastcall b_factorChange(TObject *Sender);
        void __fastcall S_dayChange(TObject *Sender);
        void __fastcall D_U_HChange(TObject *Sender);
        void __fastcall M_WChange(TObject *Sender);
private:  // User declarations
public:         // User declarations
        __fastcall TCarp(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TCarp *Carp;
//---------------------------------------------------------------------------
#endif
```

Catfish.h

```
//---------------------------------------------------------------------------
#ifndef _CatfishH
#define _CatfishH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
```

```
#include <vcl\Mask.hpp>
//---------------------------------------------------------------------------
class TCatfish : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TEdit *low_temp;
        TEdit *low_DO;
        TEdit *low_pH;
        TEdit *low_alk;
        TEdit *low_CO2;
        TEdit *low_NH4;
        TEdit *low_NO2;
        TEdit *low_hardness;
        TEdit *low_chloride;
        TEdit *low_salinity;
        TBitBtn *BitBtn1;
        TEdit *high_temp;
        TEdit *high_DO;
        TEdit *high_pH;
        TEdit *high_alk;
        TEdit *high_CO2;
        TEdit *high_NH4;
        TEdit *high_NO2;
        TEdit *high_hardness;
        TEdit *high_chloride;
        TEdit *high_salinity;
        TLabel *Label11;
        TLabel *Label12;
        TLabel *Label13;
        TLabel *Label15;
        TLabel *Label16;
        TLabel *Label17;
        TLabel *Label18;
        TLabel *Label19;
        TLabel *Label20;
        TMaskEdit *growth_factor;
        TEdit *b_factor;
        TEdit *weight_fin;
        TEdit *S_Wt;
        TEdit *S_day;
        TEdit *D_U_H;
        TLabel *_spN;
        TLabel *Label14;
        TEdit *D_O_C;
```

```
        TLabel *Label21;
        TLabel *Label22;
        TLabel *Label23;
        TEdit *M_W;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall growth_factorChange(TObject *Sender);
        void __fastcall S_WtChange(TObject *Sender);

        void __fastcall weight_finChange(TObject *Sender);
        void __fastcall b_factorChange(TObject *Sender);
        void __fastcall S_dayChange(TObject *Sender);
        void __fastcall D_U_HChange(TObject *Sender);
        void __fastcall M_WChange(TObject *Sender);
private: // User declarations
public:         // User declarations
        __fastcall TCatfish(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TCatfish *Catfish;
//---------------------------------------------------------------------------
#endif
```

_Salmon.h

```
//---------------------------------------------------------------------------
#ifndef _SalmonH
#define _SalmonH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\Mask.hpp>
//---------------------------------------------------------------------------
class TSalmon : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TEdit *low_temp;
        TEdit *low_DO;
```

```
        TEdit *low_pH;
        TEdit *low_alk;
        TEdit *low_CO2;
        TEdit *low_NH4;
        TEdit *low_NO2;
        TEdit *low_hardness;
        TEdit *low_chloride;
        TEdit *low_salinity;
        TBitBtn *BitBtn1;
        TEdit *high_temp;
        TEdit *high_DO;
        TEdit *high_pH;
        TEdit *high_alk;
        TEdit *high_CO2;
        TEdit *high_NH4;
        TEdit *high_NO2;
        TEdit *high_hardness;
        TEdit *high_chloride;
        TEdit *high_salinity;
        TLabel *Label11;
        TLabel *Label12;
        TLabel *Label13;
        TLabel *Label15;
        TLabel *Label16;
        TLabel *Label17;
        TLabel *Label18;
        TLabel *Label19;
        TLabel *Label20;
        TMaskEdit *growth_factor;
        TEdit *b_factor;
        TEdit *weight_fin;
        TEdit *S_Wt;
        TEdit *S_day;
        TEdit *D_U_H;
        TLabel *_spN;
        TLabel *Label14;
        TEdit *D_O_C;
        TLabel *Label21;
        TLabel *Label22;
        TLabel *Label23;
        TEdit *M_W;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall growth_factorChange(TObject *Sender);
        void __fastcall S_WtChange(TObject *Sender);

        void __fastcall weight_finChange(TObject *Sender);
        void __fastcall b_factorChange(TObject *Sender);
        void __fastcall S_dayChange(TObject *Sender);
        void __fastcall D_U_HChange(TObject *Sender);
        void __fastcall M_WChange(TObject *Sender);
private: // User declarations
public:         // User declarations
        __fastcall TSalmon(TComponent* Owner);
```

```
};
//---------------------------------------------------------------------------
extern TSalmon *Salmon;
//---------------------------------------------------------------------------
#endif
```

_Stripped_Bass.h

```
//---------------------------------------------------------------------------
#ifndef _Stripped_BassH
#define _Stripped_BassH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\Mask.hpp>
//---------------------------------------------------------------------------
class TStripped_Bass : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TEdit *low_temp;
        TEdit *low_DO;
        TEdit *low_pH;
        TEdit *low_alk;
        TEdit *low_CO2;
        TEdit *low_NH4;
        TEdit *low_NO2;
        TEdit *low_hardness;
        TEdit *low_chloride;
        TEdit *low_salinity;
        TBitBtn *BitBtn1;
        TEdit *high_temp;
        TEdit *high_DO;
        TEdit *high_pH;
        TEdit *high_alk;
        TEdit *high_CO2;
        TEdit *high_NH4;
        TEdit *high_NO2;
        TEdit *high_hardness;
        TEdit *high_chloride;
        TEdit *high_salinity;
```

```
        TLabel *Label11;
        TLabel *Label12;
        TLabel *Label13;
        TLabel *Label15;
        TLabel *Label16;
        TLabel *Label17;
        TLabel *Label18;
        TLabel *Label19;
        TLabel *Label20;
        TMaskEdit *growth_factor;
        TEdit *b_factor;
        TEdit *weight_fin;
        TEdit *S_Wt;
        TEdit *S_day;
        TEdit *D_U_H;
        TLabel *_spN;
        TLabel *Label14;
        TEdit *D_O_C;
        TLabel *Label21;
        TLabel *Label22;
        TLabel *Label23;
        TEdit *M_W;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall growth_factorChange(TObject *Sender);
        void __fastcall S_WtChange(TObject *Sender);

        void __fastcall weight_finChange(TObject *Sender);
        void __fastcall b_factorChange(TObject *Sender);
        void __fastcall S_dayChange(TObject *Sender);
        void __fastcall D_U_HChange(TObject *Sender);
        void __fastcall M_WChange(TObject *Sender);
private: // User declarations
public:         // User declarations
        __fastcall TStripped_Bass(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TStripped_Bass *Stripped_Bass;
//---------------------------------------------------------------------------
#endif
```

_Sturgeon.h

```
//---------------------------------------------------------------------------
#ifndef _SturgeonH
#define _SturgeonH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\Mask.hpp>
```

```
//---------------------------------------------------------------------------
class TSturgeon : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TEdit *low_temp;
        TEdit *low_DO;
        TEdit *low_pH;
        TEdit *low_alk;
        TEdit *low_CO2;
        TEdit *low_NH4;
        TEdit *low_NO2;
        TEdit *low_hardness;
        TEdit *low_chloride;
        TEdit *low_salinity;
        TBitBtn *BitBtn1;
        TEdit *high_temp;
        TEdit *high_DO;
        TEdit *high_pH;
        TEdit *high_alk;
        TEdit *high_CO2;
        TEdit *high_NH4;
        TEdit *high_NO2;
        TEdit *high_hardness;
        TEdit *high_chloride;
        TEdit *high_salinity;
        TLabel *Label11;
        TLabel *Label12;
        TLabel *Label13;
        TLabel *Label15;
        TLabel *Label16;
        TLabel *Label17;
        TLabel *Label18;
        TLabel *Label19;
        TLabel *Label20;
        TMaskEdit *growth_factor;
        TEdit *b_factor;
        TEdit *weight_fin;
        TEdit *S_Wt;
        TEdit *S_day;
        TEdit *D_U_H;
        TLabel *_spN;
        TLabel *Label14;
        TEdit *D_O_C;
        TLabel *Label21;
```

```
        TLabel *Label22;
        TLabel *Label23;
        TEdit *M_W;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall growth_factorChange(TObject *Sender);
        void __fastcall S_WtChange(TObject *Sender);

        void __fastcall weight_finChange(TObject *Sender);
        void __fastcall b_factorChange(TObject *Sender);
        void __fastcall S_dayChange(TObject *Sender);
        void __fastcall D_U_HChange(TObject *Sender);
        void __fastcall M_WChange(TObject *Sender);
private: // User declarations
public:         // User declarations
        __fastcall TSturgeon(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TSturgeon *Sturgeon;
//---------------------------------------------------------------------------
#endif
```

_Tilapia.h

```
//---------------------------------------------------------------------------
#ifndef _TilapiaH
#define _TilapiaH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\Mask.hpp>
//---------------------------------------------------------------------------
class TTilapia : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TEdit *low_temp;
        TEdit *low_DO;
        TEdit *low_pH;
        TEdit *low_alk;
        TEdit *low_CO2;
```

```
        TEdit *low_NH4;
        TEdit *low_NO2;
        TEdit *low_hardness;
        TEdit *low_chloride;
        TEdit *low_salinity;
        TBitBtn *BitBtn1;
        TEdit *high_temp;
        TEdit *high_DO;
        TEdit *high_pH;
        TEdit *high_alk;
        TEdit *high_CO2;
        TEdit *high_NH4;
        TEdit *high_NO2;
        TEdit *high_hardness;
        TEdit *high_chloride;
        TEdit *high_salinity;
        TLabel *Label11;
        TLabel *Label12;
        TLabel *Label13;
        TLabel *Label15;
        TLabel *Label16;
        TEdit *b_factor;
        TLabel *Label17;
        TEdit *weight_fin;
        TLabel *_spN;
        TLabel *Label18;
        TEdit *S_Wt;
        TLabel *Label19;
        TEdit *S_day;
        TLabel *Label20;
        TEdit *D_U_H;
        TLabel *Label14;
        TEdit *D_O_C;
        TEdit *growth_factor;
        TLabel *Label21;
        TLabel *Label22;
        TLabel *Label23;
        TEdit *M_W;
        TLabel *Label24;
        TEdit *D_A_S;
        TLabel *Label25;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall S_WtChange(TObject *Sender);
        void __fastcall growth_factorChange(TObject *Sender);

        void __fastcall D_U_HChange(TObject *Sender);
        void __fastcall weight_finChange(TObject *Sender);
        void __fastcall b_factorChange(TObject *Sender);
        void __fastcall S_dayChange(TObject *Sender);
        void __fastcall M_WChange(TObject *Sender);
        void __fastcall D_A_SChange(TObject *Sender);

private:  // User declarations
```

```
public:                 // User declarations
        __fastcall TTilapia(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TTilapia *Tilapia;
//---------------------------------------------------------------------------
#endif
```

_Tropical.h

```
//---------------------------------------------------------------------------
#ifndef _TropicalH
#define _TropicalH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\Mask.hpp>
//---------------------------------------------------------------------------
class TTropical : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TEdit *low_temp;
        TEdit *low_DO;
        TEdit *low_pH;
        TEdit *low_alk;
        TEdit *low_CO2;
        TEdit *low_NH4;
        TEdit *low_NO2;
        TEdit *low_hardness;
        TEdit *low_chloride;
        TEdit *low_salinity;
        TBitBtn *BitBtn1;
        TEdit *high_temp;
        TEdit *high_DO;
        TEdit *high_pH;
        TEdit *high_alk;
        TEdit *high_CO2;
        TEdit *high_NH4;
        TEdit *high_NO2;
        TEdit *high_hardness;
```

```
        TEdit *high_chloride;
        TEdit *high_salinity;
        TLabel *Label11;
        TLabel *Label12;
        TLabel *Label13;
        TLabel *Label15;
        TLabel *Label16;
        TLabel *Label17;
        TLabel *Label18;
        TLabel *Label19;
        TLabel *Label20;
        TMaskEdit *growth_factor;
        TEdit *b_factor;
        TEdit *weight_fin;
        TEdit *S_Wt;
        TEdit *S_day;
        TEdit *D_U_H;
        TLabel *_spN;
        TLabel *Label14;
        TEdit *D_O_C;
        TLabel *Label21;
        TLabel *Label22;
        TLabel *Label23;
        TEdit *M_W;
        void __fastcall BitBtn1Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);

        void __fastcall growth_factorChange(TObject *Sender);
        void __fastcall S_WtChange(TObject *Sender);

        void __fastcall weight_finChange(TObject *Sender);
        void __fastcall b_factorChange(TObject *Sender);
        void __fastcall S_dayChange(TObject *Sender);
        void __fastcall D_U_HChange(TObject *Sender);
        void __fastcall M_WChange(TObject *Sender);
private: // User declarations
public:         // User declarations
        __fastcall TTropical(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TTropical *Tropical;
//---------------------------------------------------------------------------
#endif
```

_Trout.h

```
//---------------------------------------------------------------------------
#ifndef _TroutH
#define _TroutH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
```

```
#include <vcl\Forms.hpp>
#include <vcl\Buttons.hpp>
//---------------------------------------------------------------------------
class TTrout : public TForm
{
__published:    // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TEdit *low_temp;
        TEdit *low_DO;
        TEdit *low_pH;
        TEdit *low_alk;
        TEdit *low_CO2;
        TEdit *low_NH4;
        TEdit *low_NO2;
        TEdit *low_hardness;
        TEdit *low_chloride;
        TEdit *low_salinity;
        TBitBtn *OK;
        TEdit *high_temp;
        TEdit *high_DO;
        TEdit *high_pH;
        TEdit *high_alk;
        TEdit *high_CO2;
        TEdit *high_NH4;
        TEdit *high_NO2;
        TEdit *high_hardness;
        TEdit *high_chloride;
        TEdit *high_salinity;
        TLabel *Label11;
        TLabel *Label12;
        TLabel *Label13;
        TLabel *Label15;
        TLabel *Label16;
        TLabel *Label17;
        TLabel *Label18;
        TLabel *Label19;
        TLabel *Label20;
        TLabel *Label14;
        TEdit *b_factor;
        TEdit *weight_fin;
        TEdit *S_Wt;
        TEdit *S_day;
        TEdit *growth_factor;
        TEdit *D_U_H;
        TEdit *D_O_C;
```

```
        TLabel *Label21;
        TLabel *Label22;
        TLabel *_spN;
        TLabel *Label23;
        TEdit *M_W;
        void __fastcall OKClick(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall growth_factorChange(TObject *Sender);
        void __fastcall S_WtChange(TObject *Sender);

        void __fastcall weight_finChange(TObject *Sender);
        void __fastcall b_factorChange(TObject *Sender);
        void __fastcall S_dayChange(TObject *Sender);
        void __fastcall D_U_HChange(TObject *Sender);
        void __fastcall M_WChange(TObject *Sender);
private:  // User declarations
public:         // User declarations
        __fastcall TTrout(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TTrout *Trout;
//---------------------------------------------------------------------------
#endif
```

Main.cpp

```
//------------------------Things to do-------------------------------------
//save to file all parameters of system and reload on opening
//reload files with waste
//start waste on opening
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "main.h"
#include "photo.h"
#include "filter_design.h"
#include "species.h"
#include "_oxygen.h"
#include "_system_design.h"
#include "_temperature.h"
#include "_feed.h"
#include "system_status.h"
#include "_Tilapia.h"
#include "growth.h"
#include "Waste.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TMain_Form *Main_Form;
```

```
extern struct AguaLog{
        int Water_Pump;
  int Air_Pump;
  int   MAN_PWM0;
  float FR_water;
  float FR_Set;
  float A_Press;
  float A_Set;
  float F_Press;
  float Filter_C_set;
  int  Sys_Water_Temp;
  int  Water_Temp_Set;
  int  Air_Temp;
  float DO;
  float DO_Set;
  float Total_Feed;
  int  FD_update_t;
  bool  L_Status;
  bool  H_Status;
  int     t_s;
  int     t_m;
  int     t_h;
  int     d_d;
  int  d_m;
  int  d_y;
  int  Update_t;
  }AD;

extern struct feed_status
{
 unsigned char agua_add:  //address of AGUA feed
 int                      AF_1;     //feeding time of feeder 1
 int                      AF_2;     //feeding time of feeder 2
 int                      AF_3;     //feeding time of feeder 3
 int                      AF_4;     //feeding time of feeder 4
 int                      AFA_1:          //time for feeder 1 to turn on (1/4 sec 0-999)
 int                      AFA_2:          //time for feeder 2 to turn on (1/4 sec 0-999)
 int                      AFA_3:          //time for feeder 3 to turn on (1/4 sec 0-999)
 int                      AFA_4:          //time for feeder 4 to turn on (1/4 sec 0-999)
 unsigned char STATUS:   //status flags for feeding done
 int                      t_s:            //second of feeding update
 int                      t_m;            //minute of feeding update
 int                      t_h;            //hour of feeding update
 int                      d_d;     //day of feeding update
 int              d_m;     //month of feeding update
}AF;

extern struct feed_log
{
 unsigned char agua_add: //address of the feeder unit
 float                    AF_cal1;        //feed delivered grams/sec
 float                    AF_cal2;        //feed delivered grams/sec
 float                    AF_cal3;        //feed delivered grams/sec
```

```
 float                  AF_cal4:          //feed delivered grams/sec
 float                  AF_Amt_D1:  //amount feed on a date
 float                  AF_Amt_D2:  //amount feed on a date
 float                  AF_Amt_D3:  //amount feed on a date
 float                  AF_Amt_D4:  //amount feed on a date
 float                  Tot_Feed_D1;//total feed delivered
 float                  Tot_Feed_D2;//total feed delivered
 float                  Tot_Feed_D3;//total feed delivered
 float                  Tot_Feed_D4;//total feed delivered
 int                    t_s:              //second of feeding
 int                    t_m:              //minute of feeding
 int                    t_h:              //hour of feeding
 int                    d_d:        //day of feeding
 int            d_m:      //month of feeding
 int                    d_y:              //year of feeding
 bool                   A_M_1:     //feeding mode true=automatic  false=manual
 bool                   A_M_2:     //feeding mode true=automatic  false=manual
 bool                   A_M_3:     //feeding mode true=automatic  false=manual
 bool                   A_M_4:     //feeding mode true=automatic  false=manual
};FL;

extern struct Waste_M
{
 float PFR;             //waste pump flow rate in gpm
 float DER;             //daily exchange rate % of total volume of system
 int  IBE;              //time interval between wasting of water from system
 int  TPO;              //maximum exchange rate allowed
 float NO3L;      //last entered nitrate level
 float NO2L;            //last entered nitrite level
 int      t_s;          //time seconds
 int  t_m;        //time min
 int      t_h;          //time hours
 int      d_d;          //date day
 int      d_m;          //date month
 int      d_y;          //date year
};W_M_S;

struct growth_data
{
  int  tank_num;      //tank number
  int  num_fish;      //number of fish in tank
       float sp_gr_fac;      //growth factor K
  float sp_b;          //b factor
  float sp_fin_wt;     //final expected wt
  float begin_wt;      //begining wt
  float present_wt;    //present weight
  float market_weight;  //market weight of fish
  float feed_rate;     //% body wieght to feed at
  int  Start_Day_age;     //days old when culture began
  int  Start_Date;     //day culture began (0-365)
  int  Grow_Out_days;  //number of days until harvest size
  int  Days_Of_Culture;  //number of days the culture
  char *sp_name;
```

```
};T[4];


//---------------------------------------------------------------------------
__fastcall TMain_Form::TMain_Form(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::BitBtn1Click(TObject *Sender)
{
        Close();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::PhotoClick(TObject *Sender)
{
        photoF->Show();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::filterClick(TObject *Sender)
{
 filter_form->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::Button3Click(TObject *Sender)
{
        select_species->Show();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::oxyClick(TObject *Sender)
{
        oxyF->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::growthClick(TObject *Sender)
{
        _growth->Show();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::Button6Click(TObject *Sender)
{
        temperature->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::Button7Click(TObject *Sender)
{
        system_design->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::Button8Click(TObject *Sender)
{
        Feed_sch->Show();
}
```

```
//---------------------------------------------------------------------------
void __fastcall TMain_Form::Button1Click(TObject *Sender)
{
        sys_status->Show();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::Button2Click(TObject *Sender)
{
Tilapia->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::Waste_TreatClick(TObject *Sender)
{
W_M->Show();
}
//---------------------------------------------------------------------------
void __fastcall TMain_Form::FormActivate(TObject *Sender)
{
        FILE *f,*g,*h,*i,*j;
            //reads from the waste status file
  h= fopen ("WASTE_STATUS.TXT", "r");
  if (h==NULL){return;}

  fscanf(h,"%f %f %d %d %f %f %d %d %d %d %d",

        &W_M_S.PFR,&W_M_S.DER,&W_M_S.IBE,&W_M_S.TPO,&W_M_S.NO3L,&W_M_S.NO2
L,
      &W_M_S.t_s,&W_M_S.t_m,&W_M_S.t_h,&W_M_S.d_d,&W_M_S.d_m,&W_M_S.d_y);
  fclose(h);

   //reads the status feed file
  f= fopen ("AGUAFEED.TXT", "r");
  if (f==NULL)return;

 fscanf(f,"%d %d %d %d %d %d %d %d %d %d %d %d %d %d",
                &AF.agua_add,
      &AF.AF_1,&AF.AF_2,&AF.AF_3,&AF.AF_4,
      &AF.AFA_1,&AF.AFA_2,&AF.AFA_3,&AF.AFA_4,
      &AF.STATUS,
      &AF.t_s,&AF.t_m,&AF.t_h,&AF.d_d,&AF.d_m);
 fclose(f);


 g= fopen ("AGUAFEED_LAST.TXT", "r");
 if (g==NULL)return;

 fscanf(g,"%d %f %f %f %f %f %f %f %f %f %f %d %d %d %d %d %d %d %d",

       &FL.agua_add,&FL.AF_cal1,&FL.AF_cal2,&FL.AF_cal3,&FL.AF_cal4,&FL.AF_Amt_D1,
      &FL.AF_Amt_D2,&FL.AF_Amt_D3,&FL.AF_Amt_D4,&FL.Tot_Feed_D1,&FL.Tot_Feed_D2,
      &FL.Tot_Feed_D3,&FL.Tot_Feed_D4,&FL.t_s,&FL.t_m,&FL.t_h,&FL.d_d,&AF.d_m,
      &FL.d_y,&FL.A_M_1,&FL.A_M_2,&FL.A_M_3,&FL.A_M_4);
fclose(g);
```

```
i= fopen ("AGUALOG_LAST.TXT", "r");
if (i==NULL)return;


  //
  //------1----2---3---4---5---6---7----8-----9--10--11----12-----13-----14-----15-----16---17--18--19---20
fscanf(i,"%d %d %d %d %d %d %d %d %f %d %d %d %f %f %f %f %d %d %d %f %d",
        &AD.t_h,              //1   hours
        &AD.t_m,              //2   minute
        &AD.t_s,              //3   second
        &AD.d_m,              //4   month
        &AD.d_d,              //5   day
        &AD.d_y,              //6   year
        &AD.L_Status,         //7   light status
        &AD.DO,               //8   dissolved oxygen  (ppm)
        &AD.Sys_Water_Temp,   //9   water temperature (degrees C)
        &AD.Water_Pump,       //10  duty cycle of water pump (%)
        &AD.Air_Pump,         //11  duty cycle of air pump (%)
        &AD.FR_water,         //12  flow rate water (gpm)
        &AD.FR_Set,           //13  flow rate set point (gpm)
        &AD.A_Press,          //14  air pressure from air pump (psi)
        &AD.A_Set,            //15  air pressure set point (psi)
        &AD.F_Press,          //16  filter differential pressure (psi)
        &AD.Update_t,         //17  time between update in AGUA con (sec)
        &AD.Water_Temp_Set,   //18  water temperature set point (degrees C)
        &AD.Air_Temp,         //19  air temperature (degrees C)
        &AD.DO_Set,           //20  dissolved oxygen set point (ppm)
        &AD.MAN_PWM0,         //21  manual PWM0 setting
        &AD.H_Status);        //22  heater status true(1) on false(0) off


fclose(i);

int k;

for(k=0;k<=3;k++)
{
 switch (k)
 {
  case 0:
  j= fopen ("GROWTH[0].TXT", "r");
  if (j==NULL)return;
  break;
  case 1:
  j= fopen ("GROWTH[1].TXT", "r");
  if (j==NULL)return;
  break;
  case 2:
  j= fopen ("GROWTH[2].TXT", "r");
  if (j==NULL)return;
  break;
  case 3:
  j= fopen ("GROWTH[3].TXT", "r");
```

```
  if (j==NULL)return;
  break;
  }

  fscanf(j,"%d %f %f %f %f %f %d %d %d %d %f %d %c",
                &T[k].tank_num,&T[k].sp_gr_fac,&T[k].sp_b,&T[k].sp_fin_wt,
      &T[k].begin_wt,&T[k].present_wt,&T[k].feed_rate,
      &T[k].Start_Day_age,&T[k].Start_Date,&T[k].Grow_Out_days,
      &T[k].Days_Of_Culture,&T[k].market_weight,&T[k].num_fish,
      &T[k].sp_name);
 fclose(j);
 }

}
//---------------------------------------------------------------------------
```

system_status.cpp

```
//---------------------------Things to do-----------------------------------
// add a band input for each parameter
// add the bands to the graphs
//
//
// Set the maximum and minimum for flow control in water and air
// Put the tie between system model and system parameters
//
// get temp to rise no greater than 1 degree per hour
//
// update data logging
// tie feeding to growth
// tie flow rate growth and feeding
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <math.h>
#include <time.h>
#include <dos.h>
#include <vcl\graphics.hpp>
#pragma hdrstop

#include "system_status.h"
#include "Graph_Output.h"
#include "pump_error.h"
//---------------------------------------------------------------------------
#pragma link "comport"
#pragma resource "*.dfm"
Tsys_status *sys_status;
extern char PRMN;
extern bool light,PRIME,FEED,D_WASTE,COND_NOTED;
extern String PRIME_N;
extern int tot_P_time;
extern int tot_F_time,FDN;
extern int T_P_D;       //time of dumping of waste (sec)
```

```
unsigned char agua_add;
bool PUMP_C=true;

enum
{
        WATER_DATA=4,
  AIR_DATA=5,
  FILTER_DATA=6,
  SYSTEM_DATA=7,
  SYS_ERR=8
};
float System_Flow_Rate,System_Flow_Setting,Pump_Level,Air_Press,Filter_Press;
int D_Time=10;
float Temp_FRC;
int MANUAL_PWM0;
struct AguaLog{
        int Water_Pump;
  int Air_Pump;
  int   MAN_PWM0;
  float FR_water;
  float FR_Set;
  float A_Press;
  float A_Set;
  float F_Press;
  float Filter_C_set;
  int  Sys_Water_Temp;
  int  Water_Temp_Set;
  int  Air_Temp;
  float DO;
  float DO_Set;
  float Total_Feed;
  int  FD_update_t;
  bool L_Status;
  bool H_Status;
  int     t_s;
  int     t_m;
  int     t_h;
  int     d_d;
  int  d_m;
  int  d_y;
  int  Update_t;
  int  Error;
  }AD,HX;

extern struct feed_status
{
 unsigned char agua_add;  //address of AGUA feed
 int                    AF_1;    //feeding time of feeder 1
 int                    AF_2;    //feeding time of feeder 2
 int                    AF_3;    //feeding time of feeder 3
 int                    AF_4;    //feeding time of feeder 4
 int                    AFA_1;          //time for feeder 1 to turn on (1/4 sec 0-999)
 int                    AFA_2;          //time for feeder 2 to turn on (1/4 sec 0-999)
 int                    AFA_3;          //time for feeder 3 to turn on (1/4 sec 0-999)
```

```
 int                    AFA_4:          //time for feeder 4 to turn on (1/4 sec 0-999)
 unsigned char STATUS:   //status flags for feeding done
 int                    t_s:            //second of feeding update
 int                    t_m:            //minute of feeding update
 int                    t_h:            //hour of feeding update
 int                    d_d:    //day of feeding
 int            d_m:     //month of feeding
}AF;

extern struct Waste_M
{
 float PFR;             //waste pump flow rate in gpm
 float DER;             //daily exchange rate % of total volume of system
 int  IBE;              //time interval between wasting of water from system
 int  TPO;              //time pump will be turned on sec
 float NO3L:    //last entered nitrate level
 float NO2L:            //last entered nitrite level
 int     t_s:           //time seconds
 int  t_m:      //time min
 int     t_h:           //time hours
 int     d_d:           //date day
 int     d_m:           //date month
 int     d_y:           //date year
}W_M_S;

extern struct growth_data
{
  int  tank_num:      //tank number
  int  num_fish:     //number of fish in tank
       float sp_gr_fac:     //growth factor K
  float sp_b:       //b factor
  float sp_fin_wt:     //final expected wt
  float begin_wt:      //begining wt
  float present_wt:     //present weight
  float market_weight:  //market weight of fish
  float feed_rate:      //% body wieght to feed at
  int  Start_Day:    //day culture was spawned
  int  Grow_Out_days;  //number of days until harvest size
  int  Days_Of_Culture:  //number of days the culture
  char* sp_name;

}T[4];

extern struct time t;
extern struct date dt;
unsigned char data_read=0;
void LogData();
void Log_Feed_Status();
void Log_Waste_Event(void);
//---------------------------------------------------------------------------
__fastcall Tsys_status::Tsys_status(TComponent* Owner)
        : TForm(Owner)
{
        ComPort->Open();
```

```
    getdate(&dt);   //get the present date (put into structure dt)
    monl->Caption=FloatToStr(dt.da_mon);
    dayl->Caption=FloatToStr(dt.da_day);
    yearl->Caption=FloatToStr(dt.da_year);
    Dump_Time->Text=IntToStr(D_Time);

}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::Get_DataTimer(TObject *Sender)
{       //getdata from system every 5 seconds

    Label11->Caption=IntToStr(T_P_D);

    ComPort->Write ("GDATA1\n");  //tell clink to get the data by CAN from AGUA con
    ComPort->Write ("GDATA2\n");
    ComPort->Write ("GDATA3\n");
    ComPort->Write ("GDATA4\n");

    ComPort->Write ("DATA4\n");   //ask clink for the data
    ComPort->Write ("DATA1\n");
    ComPort->Write ("DATA2\n");
    ComPort->Write ("DATA3\n");

   gettime(&t);    //get the present time (put into structure t)
   hourl->Caption=FloatToStr(t.ti_hour);
   minl->Caption=FloatToStr(t.ti_min);
   secl->Caption=FloatToStr(t.ti_sec);
//   Screen_Update->Enabled=true;     //!!!!!!!!!!!!!!!!!!!!!!reset
}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::Timer2Timer(TObject *Sender)
{   //logs the data every minute
   LogData();
}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::Timer3Timer(TObject *Sender)
{   //turns the lights on or off  its checked every minute
        if(light && !AD.L_Status && data_read)    //it's daylight and the controller has the light off
    {
        Light_Status->Glyph->LoadFromFile("Led1on.BMP"); //update screen
      ComPort->Write ("PHOTO_ON?\n");      //tell controller to turn on light
    }
    if(!light && AD.L_Status && data_read)       //it's night and the light is on
    {
        Light_Status->Glyph->LoadFromFile("Led1off.BMP"); //update screen
      ComPort->Write ("PHOTO_OFF?\n");      //tell controller to turn off light

    }
    if(AD.L_Status)Light_Status->Glyph->LoadFromFile("Led1on.BMP");//turn light indicator on
    else
    Light_Status->Glyph->LoadFromFile("Led1off.BMP");//turn light indicator off

}
//---------------------------------------------------------------------------
```

```
void __fastcall Tsys_status::Timer4Timer(TObject *Sender)
{  //  checks for priming and feeding of feeders every 1/2 second
   String add_agua,PT,PC,feed_code,FT,FC,prm_code;


        if(PRIME)  //if the job is to prime a feeder
   {
    if(agua_add<=9)add_agua="00"+IntToStr(agua_add); //aligns address code as string (1-9 ie. 001-009)
    else
    if(agua_add>=10 && agua_add<=99)add_agua="0"+IntToStr(agua_add);// (010-099)
    else
    add_agua=IntToStr(agua_add); //(100-127)

    if(tot_P_time<=9)PT="00"+IntToStr(tot_P_time);//aligns time code as string (1-9 ie. 001-009)
    else                                  //time is 1/4 secs
    if(tot_P_time>=10 && tot_P_time<=99)PT="0"+IntToStr(tot_P_time);// (010-099)
    else
    PT=IntToStr(tot_P_time); //(100-999)
    prm_code="0"+add_agua+IntToStr(PRMN)+PT; //net-add-feeder_num-prime_time
    PC="PRIME"+prm_code;
    ComPort->Write (PC);
    ComPort->Write ("\n");
    PRIME=false;
   }
   if(FEED)   //if the feeder flag has been turned on
   {          //aligns address code as string (1-9 ie. 001-009)
    if(agua_add<=9)add_agua="00"+IntToStr(agua_add);
    else
    if(agua_add>=10 && agua_add<=99)add_agua="0"+IntToStr(agua_add);
    else
    add_agua=IntToStr(agua_add);
            //aligns time code as string (1-9 ie. 001-009)  time is in 1/4 sec
    if(tot_F_time<=9)FT="00"+IntToStr(tot_F_time);
    else
    if(tot_F_time>=10 && tot_F_time<=99)FT="0"+IntToStr(tot_F_time);
    else
    FT=IntToStr(tot_F_time);
    Label12->Visible=true;  //show the feeding
    feed_code="0"+add_agua+IntToStr(FDN)+FT;  //net-agua_feed_add-feeder_number-feed_time
    Feed_Interval->Interval=(int)((StrToFloat(FT)/4)*1000);
    Label17->Caption=IntToStr((int)((StrToFloat(FT)/4)*1000));
    Label12->Caption="Feeding "+IntToStr(FDN);

    Feed_Interval->Enabled=true;
    FC="FEED"+feed_code;
    ComPort->Write (FC);
    ComPort->Write ("\n");
    switch(FDN)
    {
        case 1:
     AF.STATUS=AF.STATUS | 0x08; //set feed done bit
     Log_Feed_Status();//write the feeding to the log
     break;
     case 2:
```

```
      AF.STATUS=AF.STATUS | 0x04; //set feed done bit
      Log_Feed_Status();
      break;
      case 3:
      AF.STATUS=AF.STATUS | 0x02; //set feed done bit
      Log_Feed_Status();
      break;
      case 4:
      AF.STATUS=AF.STATUS | 0x01; //set feed done bit
      Log_Feed_Status();
      break;
     }
     FEED=false;      //a feeding occurred so turn off the feed flag


    }
}

//---------------------------------------------------------------------------
void __fastcall Tsys_status::M_P_DTimer(TObject *Sender)
{  //start waste pump on after 60 sec interval
   Dump_W->Caption="Dumping";
   Dump_W->Font->Color=clRed;
   M_W_T->Interval=D_Time*1000;  //set time to show dumping of waste
   M_W_T->Enabled=true;  //timer starts for waste


   ComPort->Write ("DU_W");  //send command to start dumping waste
   ComPort->Write (D_Time);
   ComPort->Write ("\n");


   M_P_D->Enabled=false;  //disable this timer
   R_P->Enabled=true;

}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::R_PTimer(TObject *Sender)
{  //resets main pump  after 10 secs
   int set;
   AD.FR_Set=Temp_FRC;
   set=(int)ceil(((Temp_FRC*(float)AD.Update_t)/60)*1714);
   Label23->Font->Color=clBlue;
   Label23->Caption=FloatToStr(Temp_FRC);
   ComPort->Write ("FR_C");
   ComPort->Write (set);
   ComPort->Write ("\n");
   R_P->Enabled=false;

}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::Waste_TimerTimer(TObject *Sender)
{   //checks for dumping of waste  every 1 sec
        int set;
```

```
  int temp_TPO;

        if(D_WASTE)  //if the waste flag is set to true dump waste
    {
      Waste_Timer->Enabled=false;
      Temp_FRC=AD.FR_Set;
        D_Time=W_M_S.TPO;
        Log_Waste_Event();
        M_W_T->Interval=D_Time*1000;
        ComPort->Write ("WC_C");     // turns off the main system pump
                                //disable control system
        ComPort->Write ("\n");
        M_P_D->Enabled=true;   //start 60 sec timer to start dumping waste
    }
}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::M_W_TTimer(TObject *Sender)
{
  Dump_W->Caption="Dump Waste";
  Dump_W->Font->Color=clBlack;
  M_W_T->Enabled=false;
  Waste_Timer->Enabled=true;
  D_WASTE=false;
}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::Screen_UpdateTimer(TObject *Sender)
{
  int whole;
  float tot,frac;

  AD.Water_Pump=(int)(((float)HX.Water_Pump/256.0)*100.0);       //gets the pumping rate in % of duty
cycle
  AD.FR_water=(((float)(HX.FR_water/HX.Update_t))*60)/1714; // pulses/sec=pulses/sample_time
  sys_flow->Text=FloatToStr(AD.FR_water);              //GPM=1714 pulses/min
  AD.FR_Set=((float)(HX.FR_Set/HX.Update_t)*60)/1714;
  sys_set->Text=FloatToStr(AD.FR_Set); //gets the system setting of flow rate

  AD.Air_Pump=(int)(((float)HX.Air_Pump/256.0)*100.0);
  AD.A_Press=((float)HX.A_Press*.015738)-.6446;          // get air pressure 450 mV/kPa   4.5v full scale
span
  air_P->Text=FloatToStr(AD.A_Press);
  AD.A_Set=((float)HX.A_Set*.015738)-.6446;
  air_set->Text=FloatToStr(AD.A_Set);
  tot=((float)HX.DO/204.0)*14.0;  //reported DO
  whole=(int)tot;
  frac=((float)((int)((tot-(float)whole)*10)))/10.0;
  AD.DO=(float)whole+frac;
  O2_level->Text=FloatToStr(AD.DO);

  tot=((float)HX.DO_Set/204.8)*14;
  whole=(int)tot;
  frac=((float)((int)((tot-(float)whole)*10)))/10.0;
  AD.DO_Set=(float)whole+frac;
```

```
 set_DO->Text=FloatToStr(AD.DO_Set); //this needs to be finished

 AD.F_Press=((float)HX.F_Press*.015738)-.6446;     //gets filter differential pressure 45 mV/kPa  4.5v
full scale span
 filter_P->Text=FloatToStr(AD.F_Press);
 AD.Filter_C_set= ((float)HX.Filter_C_set*.015738)-.6446;
 fil_set_P->Text=FloatToStr(AD.Filter_C_set);

 AD.Sys_Water_Temp=HX.Sys_Water_Temp;
 Water_Temp->Text=IntToStr(AD.Sys_Water_Temp);
 AD.Total_Feed=HX.Total_Feed;    //needs to be finished
 if(HX.L_Status==0)AD.L_Status=false;  //gets the light status light is off
 else
 AD.L_Status=true;            //light is on
 AD.Update_t=HX.Update_t;                //how often the microcontroller updates its controls
 AD.Air_Temp=HX.Air_Temp;                //air temperature
 A_temp->Text=IntToStr(AD.Air_Temp);
 AD.Water_Temp_Set=HX.Water_Temp_Set;         //set point for water temperature
 W_T_Set->Text=IntToStr(AD.Water_Temp_Set);

 if(HX.H_Status==0)Heat_Status->Glyph->LoadFromFile("Led1off.BMP");
 else Heat_Status->Glyph->LoadFromFile("Led1on.BMP");

 switch(HX.Error)
  {
   case 1:
    if(!COND_NOTED)
    PUMP_ERR->ShowModal();
   break;
   case 2:
    send_PWM0_data(MANUAL_PWM0);
   break;
   case 3:
    PUMP_C=false;
    P_Man->Visible=true;
    P_Man_V->Visible=true;
    PUMP_CONTROL->Caption="MANUAL";
   break;
  }

     pwm_level->Text=IntToStr(AD.Water_Pump);
     A_P_L->Text=IntToStr(AD.Air_Pump);

     data_read=1;
}
//---------------------------------------------------------------------------

void __fastcall Tsys_status::FormClose(TObject *Sender, TCloseAction &Action)
{
        ComPort->Close();
}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::ComPortWriteByte(TObject *Sender,
        char &ReceivedByte)
```

```
{
        if (ReceivedByte == '\n')
                ReceivedByte = '\r';
}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::ComPortReadLine(TObject *Sender,
      char *ReceivedLine)
{   //collects the data from the stystem

  int data_code,datab1,datab2,datab3,datab4,datab5,datab6,datab7,datab8;
    //data_code = 4-8 4=water,5=air,6=filter,7=system 8=error
                StatusBar->SimpleText= ReceivedLine;
    sscanf (ReceivedLine , "%2x %2x %2x %2x %2x %2x %2x %2x %2x",
        &data_code,&datab1,&datab2,&datab3,&datab4,&datab5,&datab6,&datab7,&datab8);
    switch(data_code)
    {
      case WATER_DATA:
        HX.Water_Pump=datab1;
        HX.FR_water=(datab2<<8)|datab3;
        HX.FR_Set=datab4;
       HX.MAN_PWM0=datab5;
      break;

      case AIR_DATA:
       HX.Air_Pump=datab1;
       HX.A_Press=(datab2<<8)|datab3;
       HX.A_Set=datab4;
       HX.DO=datab5;
       HX.DO_Set=datab6;
      break;

      case FILTER_DATA:
        HX.F_Press=(datab1<<8)|datab2;
       HX.Filter_C_set=(datab3<<8)|datab4;
      break;

      case SYSTEM_DATA:
        HX.Total_Feed=datab1;
       HX.L_Status=datab2;
       HX.Update_t=datab3;
       HX.FD_update_t=datab4;
       HX.Sys_Water_Temp=datab5;
       HX.Air_Temp=datab6;
       HX.Water_Temp_Set=datab7;
       HX.H_Status=datab8;
      break;

      case SYS_ERR:
       HX.Error=datab1;
      break;

    }
```

```
}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::Light_StatusClick(TObject *Sender)
{
  if(AD.L_Status)
  {
   ComPort->Write ("PHOTO_OFF?\n");
   Light_Status->Glyph->LoadFromFile("Led1off.BMP"); //update screen
  }
  else
  {
   ComPort->Write ("PHOTO_ON?\n");
   Light_Status->Glyph->LoadFromFile("Led1on.BMP"); //update screen
  }

}
//---------------------------------------------------------------------------

void __fastcall Tsys_status::Sys_Set_FRClick(TObject *Sender)
{            //changes the flow rate of the water in the system
        int set;
  pwm_level->MaxLength = 3;
        set=ceil((((atof(Tsys_status::sys_reset->Text.c_str())*AD.Update_t)/60.0)*1714.0));
  AD.FR_Set=atof(Tsys_status::sys_reset->Text.c_str());
  Label23->Caption=FloatToStr(AD.FR_Set);
  ComPort->Write ("FR_C");
  ComPort->Write (set);
  ComPort->Write ("\n");
}
//---------------------------------------------------------------------------

void __fastcall Tsys_status::Air_ChangeClick(TObject *Sender)
{           //changes the air pressure in the system
        int set;

          AD.A_Set=floor(((atof(Tsys_status::Air_Reset->Text.c_str()))+.6446)/.015738);
  set=(int)AD.A_Set;
  ComPort->Write ("FR_A");
  ComPort->Write (set);
  ComPort->Write ("\n");
}
//---------------------------------------------------------------------------

void __fastcall Tsys_status::reset_DOClick(TObject *Sender)
{
 int set;

          AD.DO_Set=floor(((atof(Tsys_status::DO_reset->Text.c_str()))/14.0)*204.8);
  set=(int)AD.DO_Set;
  ComPort->Write ("DO_A");
  ComPort->Write (set);
  ComPort->Write ("\n");
```

```
}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::Temp_ResetClick(TObject *Sender)
{
  int set;

        AD.Water_Temp_Set=atoi(Tsys_status::T_Reset->Text.c_str());
  set=AD.Water_Temp_Set;
  ComPort->Write ("SET_T");
  ComPort->Write (set);
  ComPort->Write ("\n");
}
//---------------------------------------------------------------------------

void LogData()
{           //logs the data from the system


        FILE *f,*g;
  f= fopen ("AGUALOG.TXT", "at");
  if (f==NULL)return;

  TDateTime dtPresent = Now();
  Word wYear, wMonth, wDay;
  Word wHour, wMinute, wSecond, wMillisecond;
  DecodeDate(dtPresent, wYear, wMonth, wDay);
  DecodeTime(dtPresent, wHour, wMinute, wSecond, wMillisecond);

  AD.Total_Feed =0;

  AD.t_s = wSecond;
  AD.t_m = wMinute;
  AD.t_h = wHour;
  AD.d_d = wDay;
  AD.d_m = wMonth;
  AD.d_y = wYear;
    //
    //------1----2---3---4---5---6---7----8-----9--10--11----12-----13-----14-----15-----16---17--18--19---20

fprintf(f,"%d\t%d\t%d\t%d\t%d\t%d\t%d\t%4.2f\t%d\t%d\t%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t
%d\t%d\t%4.2f\t",
          AD.t_h,           //1   hours
          AD.t_m,           //2   minute
          AD.t_s,           //3   second
          AD.d_m,           //4   month
          AD.d_d,           //5   day
          AD.d_y,           //6   year
          AD.L_Status,      //7   light status
          AD.DO,            //8   dissolved oxygen  (ppm)
          AD.Sys_Water_Temp,    //9   water temperature (degrees C)
          AD.Water_Pump,        //10  duty cycle of water pump (%)
          AD.Air_Pump,          //11  duty cycle of air pump (%)
          AD.FR_water,          //12  flow rate water (gpm)
          AD.FR_Set,            //13  flow rate set point (gpm)
```

```
        AD.A_Press.          //14  air pressure from air pump (psi)
        AD.A_Set.            //15  air pressure set point (psi)
        AD.F_Press.          //16  filter differential pressure (psi)
        AD.Update_t.         //17  time between update in AGUA con (sec)
        AD.Water_Temp_Set.   //18  water temperature set point (degrees C)
        AD.Air_Temp.         //19  air temperature (degrees C)
        AD.DO_Set);          //20  dissolved oxygen set point (ppm)
  fprintf(f, "\n");
  fclose(f);

  g= fopen ("AGUALOG_LAST.TXT", "w");
  if (g==NULL)return;
   //
   //------1----2---3---4---5---6---7----8-----9--10--11----12-----13-----14-----15-----16---17--18--19---20

fprintf(g,"%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%4.2f\t%d\t%d\t%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t
%d\t%d\t%4.2f\t",
        AD.t_h,              //1   hours
        AD.t_m,              //2   minute
        AD.t_s,              //3   second
        AD.d_m,              //4   month
        AD.d_d,              //5   day
        AD.d_y,              //6   year
        AD.L_Status,         //7   light status
        AD.DO,               //8   dissolved oxygen  (ppm)
        AD.Sys_Water_Temp,   //9   water temperature (degrees C)
        AD.Water_Pump,       //10  duty cycle of water pump (%)
        AD.Air_Pump,         //11  duty cycle of air pump (%)
        AD.FR_water,         //12  flow rate water (gpm)
        AD.FR_Set,           //13  flow rate set point (gpm)
        AD.A_Press,          //14  air pressure from air pump (psi)
        AD.A_Set,            //15  air pressure set point (psi)
        AD.F_Press,          //16  filter differental pressure (psi)
        AD.Update_t,         //17  time between update in AGUA con (sec)
        AD.Water_Temp_Set,   //18  water temperature set point (degrees C)
        AD.Air_Temp,         //19  air temperature (degrees C)
        AD.DO_Set);          //20  dissolved oxygen set point (ppm)
  fclose(g);


}


//---------------------------------------------------------------------------
void Log_Feed_Status(void)
{
        FILE * f;
  f= fopen ("AGUAFEED.TXT", "w");
  if (f==NULL)return;

  gettime(&t);
  getdate(&dt);
  AF.t_s = t.ti_sec;
  AF.t_m = t.ti_min;
```

```
  AF.t_h = t.ti_hour;
  AF.d_d = dt.da_day;
  AF.d_m = dt.da_mon;

  fprintf(f,"%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",
            AF.agua_add,                    //address
      AF.AF_1,AF.AF_2,AF.AF_3,AF.AF_4,      //time to feed
      AF.AFA_1,AF.AFA_2,AF.AFA_3,AF.AFA_4,  //amount to feed
      AF.STATUS,                            //status flags
      AF.t_s,AF.t_m,AF.t_h,AF.d_d,AF.d_m);  //time
  fprintf(f, "\n");
  fclose(f);


}

void __fastcall Tsys_status::Dump_WClick(TObject *Sender)
{ //manually turn on the waste pump
  int temp_TPO;
  Temp_FRC=AD.FR_Set;
  D_Time=atoi(Tsys_status::Dump_Time->Text.c_str());
  temp_TPO=W_M_S.TPO;  //temp store Time Pump On
  W_M_S.TPO=D_Time;    //for logging
  Log_Waste_Evcnt();
  W_M_S.TPO=temp_TPO;  //resets the event to auto value
  M_W_T->Interval=D_Time*1000;


  Waste_Timer->Enabled=false;

  ComPort->Write ("WC_C");     // turns off the main system pump
                               //disable control system
  ComPort->Write ("\n");
  M_P_D->Enabled=true;   //start 30 sec timer to start dumping waste

}
//---------------------------------------------------------------------------

void __fastcall Tsys_status::VisualClick(TObject *Sender)
{
        Visual_Output->Show();
}
//---------------------------------------------------------------------------


void __fastcall Tsys_status::Feed_IntervalTimer(TObject *Sender)
{
  Label12->Visible=false;


  Feed_Interval->Enabled=false;
}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::P_ManClick(TObject *Sender)
```

```
{
 MANUAL_PWM0=atoi(Tsys_status::P_Man_V->Text.c_str());
 send_PWM0_data(MANUAL_PWM0);
}
//---------------------------------------------------------------------------
void __fastcall Tsys_status::PUMP_CONTROLClick(TObject *Sender)
{
        if(PUMP_C)
  {
        PUMP_C=false;
    P_Man->Visible=true;
        P_Man_V->Visible=true;
    PUMP_CONTROL->Caption="MANUAL";
        ComPort->Write ("MCT");   //manual control is 0
        ComPort->Write ("\n");
    Label23->Caption="man";
  }
  else
 if(!PUMP_C)
  {
        PUMP_C=true;
    P_Man->Visible=false;
        P_Man_V->Visible=false;
    PUMP_CONTROL->Caption="AUTO";
        ComPort->Write ("MCF");   //manual control is 1
        ComPort->Write ("\n");
    Label23->Caption="auto";
  }
}
//---------------------------------------------------------------------------

void __fastcall Tsys_status::send_PWM0_data(int PWM0)
{
 ComPort->Write ("MC");
 ComPort->Write (PWM0);
 ComPort->Write ("\n");
}

//---------------------------------------------------------------------------

void Log_Waste_Event(void)
{

  gettime(&t);
  getdate(&dt);
  W_M_S.t_s = t.ti_sec;
  W_M_S.t_m = t.ti_min;
  W_M_S.t_h = t.ti_hour;
  W_M_S.d_d = dt.da_day;
  W_M_S.d_m = dt.da_mon;
  W_M_S.d_y = dt.da_year;

  FILE * h;
   h= fopen ("WASTE_EVENT.TXT", "at");
```

```
  if (h==NULL)return;
//   fprintf(h,"scc min hrs day mon year PFR  DER  TBW PTOT NO3 NO2 \n");
  fprintf(h,"%d\t%d\t%d\t%d\t%d\t%d\t%4.2f\t%4.2f\t%d\t%d\t%4.2f\t%4.2f\t\n",
        W_M_S.t_s,W_M_S.t_m,W_M_S.t_h,W_M_S.d_d,W_M_S.d_m,W_M_S.d_y,
        W_M_S.PFR,W_M_S.DER,W_M_S.IBE,W_M_S.TPO,W_M_S.NO3L,W_M_S.NO2L);

  fclose(h);


}

//---------------------------------------------------------------------------
void __fastcall Tsys_status::FormActivate(TObject *Sender)
{

        int set;

  //update the system to the last settings used
                //set flow rate of water
  sys_set->Text=FloatToStr(AD.FR_Set); //gets the system setting of flow rate
  set=(int)floor((((AD.FR_Set*AD.Updatc_t)/60.0)*1714.0));
  ComPort->Write ("FR_C");
  ComPort->Write (set);
  ComPort->Write ("\n");
        //set the DO level
  set_DO->Text=FloatToStr(AD.DO_Set);
  set=(int)floor((AD.DO_Set/14.0)*204.8);;
  ComPort->Write ("DO_A");
  ComPort->Write (set);
  ComPort->Write ("\n");
        //set temperature setpoint
  W_T_Set->Text=IntToStr(AD.Water_Temp_Set);
  set=AD.Water_Temp_Set;
  ComPort->Write ("SET_T");
  ComPort->Write (set);
  ComPort->Write ("\n");
}
//---------------------------------------------------------------------------
```

growth.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include <math.h>
#include "growth.h"
//---------------------------------------------------------------------------
#pragma link "ChartFX"

#pragma resource "*.dfm"
```

```
T_growth *_growth;

extern int day;
extern struct growth_data
{
   int   tank_num;       //tank number
   int   num_fish;       //number of fish in tank
           float sp_gr_fac;      //growth factor K
   float sp_b;           //b factor
   float sp_fin_wt;      //final expected wt
   float begin_wt;       //begining wt
   float present_wt;     //present weight
   float market_weight;  //market weight of fish
   float feed_rate;      //% body wieght to feed at
   int   Start_Day_age;     //days old when culture began
   int   Start_Date;     //day culture began (0-365)
   int   Grow_Out_days;  //number of days until harvest size
   int   Days_Of_Culture;  //number of days the culture
   char* sp_name;

}T[4];

void Store_Growth_Data(int);
//---------------------------------------------------------------------------
__fastcall T_growth::T_growth(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall T_growth::Button1Click(TObject *Sender)
{  //reruns the calculations
 int i;

 switch(Tanks->PageIndex)
 {
 case 0:

                  Edit1->Text=FloatToStr(T[0].sp_fin_wt);
     Edit2->Text=FloatToStr(T[0].sp_gr_fac);
     Edit3->Text=FloatToStr(T[0].sp_b);
     Edit4->Text=FloatToStr(T[0].Days_Of_Culture);
   T1_ChartFX->OpenDataEx(COD_VALUES,1,T[0].Grow_Out_days+T[0].Days_Of_Culture-1);


     for (i=0;i<=T[0].Grow_Out_days+T[0].Days_Of_Culture-1;i++)        //T[Tanks-
>PageIndex].Grow_Out_days-1
     {
       T1_ChartFX->ThisSerie= 0;
       T1_ChartFX->Value[i]= T[0].sp_fin_wt * pow((1 - exp(-T[0].sp_gr_fac * i/365
)),T[0].sp_b);//{sets the Y value}

       T1_ChartFX->XValue[i]= i;     // {sets the X value}
     }
    T1_ChartFX->CloseData(COD_VALUES);
```

```
      T1_W_C->Text= FloatToStr(T[0].sp_fin_wt * pow((1 - exp(-T[0].sp_gr_fac *
(float)(T[0].Days_Of_Culture)/365 )),T[0].sp_b));

    break;

 case 1:
    T2_ChartFX->OpenDataEx(COD_VALUES,1,T[1].Grow_Out_days+T[1].Days_Of_Culture-1);


      for (i=0;i<=T[1].Grow_Out_days+T[1].Days_Of_Culture-1;i++)
       {
         T2_ChartFX->ThisSerie= 0;
         T2_ChartFX->Value[i]= T[1].sp_fin_wt * pow((1 - exp(-T[1].sp_gr_fac * i/365
)),T[1].sp_b);//{sets the Y value}

         T2_ChartFX->XValue[i]= i;      // {sets the X value}
       }
      T2_ChartFX->CloseData(COD_VALUES);
      T2_W_C->Text= FloatToStr(T[1].sp_fin_wt * pow((1 - exp(-T[1].sp_gr_fac *
(float)(T[1].Days_Of_Culture)/365 )),T[1].sp_b));
      Edit5->Text=IntToStr(T[1].sp_fin_wt);
    break;

 case 2:
    T3_ChartFX->OpenDataEx(COD_VALUES,1,T[2].Grow_Out_days+T[2].Days_Of_Culture-1);


      for (i=0;i<=T[2].Grow_Out_days+T[2].Days_Of_Culture-1;i++)
       {
         T3_ChartFX->ThisSerie= 0;
         T3_ChartFX->Value[i]= T[2].sp_fin_wt * pow((1 - exp(-T[2].sp_gr_fac * i/365
)),T[2].sp_b);//{sets the Y value}

         T3_ChartFX->XValue[i]= i;      // {sets the X value}
       }
      T3_ChartFX->CloseData(COD_VALUES);
      T3_W_C->Text= FloatToStr(T[2].sp_fin_wt * pow((1 - exp(-T[2].sp_gr_fac *
(float)(T[2].Days_Of_Culture)/365 )),T[2].sp_b));
    break;

 case 3:
    T4_ChartFX->OpenDataEx(COD_VALUES,1,T[3].Grow_Out_days+T[3].Days_Of_Culture-1);


      for (i=0;i<=T[3].Grow_Out_days+T[3].Days_Of_Culture-1;i++)
       {
         T4_ChartFX->ThisSerie= 0;
         T4_ChartFX->Value[i]= T[3].sp_fin_wt * pow((1 - exp(-T[3].sp_gr_fac * i/365
)),T[3].sp_b);//{sets the Y value}

         T4_ChartFX->XValue[i]= i;      // {sets the X value}
       }
      T4_ChartFX->CloseData(COD_VALUES);
      T4_W_C->Text= FloatToStr(T[3].sp_fin_wt * pow((1 - exp(-T[3].sp_gr_fac *
```

```
(float)(T[3].Days_Of_Culture)/365 )),T[3].sp_b));
    break;
  }

}
//---------------------------------------------------------------------------
void __fastcall T_growth::T4_1_TChange(TObject *Sender)
{  //calcs the number of days left until harvest weight is reached

  T[Tanks->PageIndex].Grow_Out_days=atoi(T_growth::T4_1_T->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::SpeedButton1Click(TObject *Sender)
{
   TDateTime Date;
         DateSeparator = '/';
         Label7->Caption = DateToStr(Date);

}
//---------------------------------------------------------------------------



void __fastcall T_growth::T2_1_TChange(TObject *Sender)
{
 T[1].Grow_Out_days=atoi(T_growth::T2_1_T->Text.c_str());
}
//---------------------------------------------------------------------------

void __fastcall T_growth::T1_1_TChange(TObject *Sender)
{
T[0].Grow_Out_days=atoi(T_growth::T1_1_T->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T1_wtChange(TObject *Sender)
{
 T[0].sp_fin_wt=atof(T_growth::T1_wt->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T1_S_DChange(TObject *Sender)
{
 T[0].Start_Date=atoi(T_growth::T1_S_D->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T1_grw_fChange(TObject *Sender)
{
 T[0].sp_gr_fac=atof(T_growth::T1_grw_f->Text.c_str());
 T1_W_C->Text=T[0].present_wt=(T[0].sp_fin_wt * pow((1 - exp(-T[0].sp_gr_fac *
(float)(T[0].Days_Of_Culture)/365 )),T[0].sp_b));

}
//---------------------------------------------------------------------------
void __fastcall T_growth::T1_b_facChange(TObject *Sender)
{
```

```
 T[0].sp_b=atof(T_growth::T1_b_fac->Text.c_str());
 T1_W_C->Text=T[0].present_wt=(T[0].sp_fin_wt * pow((1 - exp(-T[0].sp_gr_fac *
(float)(T[0].Days_Of_Culture)/365 )),T[0].sp_b));

}
//---------------------------------------------------------------------------
void __fastcall T_growth::T1_S_WChange(TObject *Sender)
{
 T[0].begin_wt=atof(T_growth::T1_S_W->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T1_D_O_CChange(TObject *Sender)
{
 T[0].Days_Of_Culture=atof(T_growth::T1_D_O_C->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T4_b_facChange(TObject *Sender)
{
  T[3].sp_b=atof(T_growth::T4_b_fac->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T4_grw_fChange(TObject *Sender)
{
  T[3].sp_fin_wt=atof(T_growth::T4_wt->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T4_wtChange(TObject *Sender)
{
   T[3].sp_fin_wt=atof(T_growth::T4_wt->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T4_S_DChange(TObject *Sender)
{
  T[3].Start_Date=atoi(T_growth::T4_S_D->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T4_S_WChange(TObject *Sender)
{
  T[0].begin_wt=atof(T_growth::T4_S_W->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T3_grw_fChange(TObject *Sender)
{
  T[2].sp_fin_wt=atof(T_growth::T3_wt->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T3_b_facChange(TObject *Sender)
{
  T[2].sp_b=atof(T_growth::T3_b_fac->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T3_D_O_CChange(TObject *Sender)
{
  T[2].Days_Of_Culture=atof(T_growth::T3_D_O_C->Text.c_str());
```

```
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T3_S_DChange(TObject *Sender)
{
 T[1].Start_Date=atoi(T_growth::T3_S_D->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T3_S_WChange(TObject *Sender)
{
 T[2].begin_wt=atof(T_growth::T3_S_W->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T3_wtChange(TObject *Sender)
{
  T[2].sp_fin_wt=atof(T_growth::T3_wt->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T2_wtChange(TObject *Sender)
{
 T[1].sp_fin_wt=atof(T_growth::T2_wt->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T2_grw_fChange(TObject *Sender)
{
 T[1].sp_fin_wt=atof(T_growth::T2_wt->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T2_b_facChange(TObject *Sender)
{
 T[1].sp_b=atof(T_growth::T2_b_fac->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T2_S_DChange(TObject *Sender)
{
 T[1].Start_Date=atoi(T_growth::T3_S_D->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T2_S_WChange(TObject *Sender)
{
 T[1].begin_wt=atof(T_growth::T3_S_W->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T2_StoreClick(TObject *Sender)
{
 Store_Growth_Data(1);
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T1_StoreClick(TObject *Sender)
{
 Store_Growth_Data(0);
}
//---------------------------------------------------------------------------

void __fastcall T_growth::T3_StoreClick(TObject *Sender)
```

```
{
 Store_Growth_Data(2);
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T4_StoreClick(TObject *Sender)
{
 Store_Growth_Data(3);
}
//---------------------------------------------------------------------------
void __fastcall T_growth::T2_D_O_CChange(TObject *Sender)
{
 T[1].Days_Of_Culture=atof(T_growth::T2_D_O_C->Text.c_str());
}
//---------------------------------------------------------------------------

void __fastcall T_growth::T4_D_O_CChange(TObject *Sender)
{
 T[3].Days_Of_Culture=atof(T_growth::T4_D_O_C->Text.c_str());
}

//---------------------------------------------------------------------------

void Store_Growth_Data(int i)
{

  FILE *j;


{
  switch (i)
  {
        case 0:
    j= fopen ("GROWTH[0].TXT", "w");
    if (j==NULL)return;
    break;
  case 1:
    j= fopen ("GROWTH[1].TXT", "w");
    if (j==NULL)return;
    break;
  case 2:
    j= fopen ("GROWTH[2].TXT", "w");
    if (j==NULL)return;
    break;
  case 3:
    j= fopen ("GROWTH[3].TXT", "w");
    if (j==NULL)return;
    break;
  }

  fprintf(j,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t%d\t%d\t%d\t%4.2f\t%d\t%c\t",
                T[i].tank_num,T[i].sp_gr_fac,T[i].sp_b,
      T[i].sp_fin_wt,T[i].begin_wt,T[i].present_wt,
      T[i].feed_rate,T[i].Start_Day_age,T[i].Start_Date.T[i].Grow_Out_days,
      T[i].Days_Of_Culture,T[i].market_weight,
```

```
        T[i].num_fish,T[i].sp_name);
    fclose(j);
  }

 }

//---------------------------------------------------------------------------
void __fastcall T_growth::TanksEnter(TObject *Sender)
{

    int i,j,k,temp;
    FILE *f;

  for(k=0;k<=3;k++)
  {
  switch (k)
  {
  case 0:
  f= fopen ("GROWTH[0].TXT", "r");
  if (f==NULL)return;
  break;
  case 1:
  f= fopen ("GROWTH[1].TXT", "r");
  if (f==NULL)return;
  break;
  case 2:
  f= fopen ("GROWTH[2].TXT", "r");
  if (f==NULL)return;
  break;
  case 3:
  f= fopen ("GROWTH[3].TXT", "r");
  if (f==NULL)return;
  break;
  }

  fscanf(f,"%d %f %f %f %f %f %d %d %d %d %f %d %c",
                &T[k].tank_num,&T[k].sp_gr_fac,&T[k].sp_b,&T[k].sp_fin_wt,
       &T[k].begin_wt,&T[k].present_wt,&T[k].feed_rate,
       &T[k].Start_Day_age,&T[k].Start_Date,&T[k].Grow_Out_days,
       &T[k].Days_Of_Culture,&T[k].market_weight,&T[k].num_fish,
       &T[k].sp_name);
  fclose(f);
  }

/*   T1_sp_N->Caption=T[0].sp_name;      //title of page
   T2_sp_name->Caption=T[1].sp_name;
  T3_sp_name->Caption=T[2].sp_name;
  T4_sp_name->Caption=T[3].sp_name;   */

        T1_wt->Text=FloatToStr(T[0].sp_fin_wt); //final expected weight
        T2_wt->Text=FloatToStr(T[1].sp_fin_wt);
        T3_wt->Text=FloatToStr(T[2].sp_fin_wt);
  T4_wt->Text=FloatToStr(T[3].sp_fin_wt);
```

```
T1_l_T->Text=IntToStr(T[0].Grow_Out_days); //days till harvest
T2_l_T->Text=IntToStr(T[1].Grow_Out_days);
T3_l_T->Text=IntToStr(T[2].Grow_Out_days);
T4_l_T->Text=IntToStr(T[3].Grow_Out_days);

T1_S_D->Text=IntToStr(T[0].Start_Date);   //date culture started
T2_S_D->Text=IntToStr(T[1].Start_Date);
T3_S_D->Text=IntToStr(T[2].Start_Date);
T4_S_D->Text=IntToStr(T[3].Start_Date);


T1_wt->Text=FloatToStr(T[0].sp_fin_wt);  //final weight of fish
T2_wt->Text=FloatToStr(T[1].sp_fin_wt);
T3_wt->Text=FloatToStr(T[2].sp_fin_wt);
T4_wt->Text=FloatToStr(T[3].sp_fin_wt);


Label9->Caption=IntToStr(T[0].Days_Of_Culture);
temp=T[0].Days_Of_Culture=T[0].Start_Day_age+(day-T[0].Start_Date);   //calculate day of culture
T[1].Days_Of_Culture=T[1].Start_Day_age+(day-T[1].Start_Date);
T[2].Days_Of_Culture=T[2].Start_Day_age+(day-T[2].Start_Date);
T[3].Days_Of_Culture=T[3].Start_Day_age+(day-T[3].Start_Date);


T1_D_O_C->Text=IntToStr(T[0].Days_Of_Culture); //days of culture
T2_D_O_C->Text=IntToStr(T[1].Days_Of_Culture);
T3_D_O_C->Text=IntToStr(T[2].Days_Of_Culture);
T4_D_O_C->Text=IntToStr(T[3].Days_Of_Culture);

T1_grw_f->Text=FloatToStr(T[0].sp_gr_fac); //growth factor for culture
T2_grw_f->Text=FloatToStr(T[1].sp_gr_fac);
T3_grw_f->Text=FloatToStr(T[2].sp_gr_fac);
T4_grw_f->Text=FloatToStr(T[3].sp_gr_fac);

  Edit1->Text=IntToStr(day);
  Edit2->Text=FloatToStr(T[0].sp_gr_fac);
  Edit3->Text=FloatToStr(T[0].sp_b);


T1_b_fac->Text=FloatToStr(T[0].sp_b);  //b factor for culture
T2_b_fac->Text=FloatToStr(T[1].sp_b);
T3_b_fac->Text=FloatToStr(T[2].sp_b);
T4_b_fac->Text=FloatToStr(T[3].sp_b);


T1_W_C->Text=FloatToStr(T[0].present_wt); //weight at culture day
T2_W_C->Text=FloatToStr(T[1].present_wt);
T3_W_C->Text=FloatToStr(T[2].present_wt);
T4_W_C->Text=FloatToStr(T[3].present_wt);



T1_S_W->Text=FloatToStr(T[0].begin_wt); //starting weight for culture
T2_S_W->Text=FloatToStr(T[1].begin_wt);
```

```
   T3_S_W->Text=FloatToStr(T[2].begin_wt);
   T4_S_W->Text=FloatToStr(T[3].begin_wt);
   T[0].Days_Of_Culture=temp;
   Edit4->Text=IntToStr(T[0].Days_Of_Culture);
 T1_W_C->Text= T[0].present_wt=(T[0].sp_fin_wt * pow((1 - (exp(-T[0].sp_gr_fac *
((float)(T[0].Days_Of_Culture)/365.0 )))),T[0].sp_b));


for(j=0;j<=3;j++)
{

 switch(j)
 {

          case 0:



     if(T[0].sp_fin_wt==0 || T[0].sp_gr_fac==0 || T[0].sp_b==0)break;   //not enough data to calculate
     T1_ChartFX->OpenDataEx(COD_VALUES,1,T[0].Grow_Out_days+T[0].Days_Of_Culture);

      for (i=0;i<=T[0].Grow_Out_days+T[0].Days_Of_Culture-1;i++)
       {
         T1_ChartFX->ThisSerie= 0;
         T1_ChartFX->Value[i]= T[0].sp_fin_wt * pow((1 - exp(-T[0].sp_gr_fac * i/365
)),T[0].sp_b);//{sets the Y value}
         T1_ChartFX->XValue[i]= i;     // {sets the X value}
       }
     T1_ChartFX->CloseData(COD_VALUES);
  //     T1_W_C->Text= FloatToStr(T[0].sp_fin_wt * pow((1 - exp(-T[0].sp_gr_fac *
(float)(T[0].Days_Of_Culture)/365 )),T[0].sp_b));

  //     Edit4->Text=FloatToStr(T[0].sp_fin_wt * pow((1 - exp(-T[0].sp_gr_fac *
(float)(T[0].Days_Of_Culture)/365 )),T[0].sp_b));
      break;

   case 1:
     if(T[1].sp_fin_wt==0 || T[1].sp_gr_fac==0 || T[1].sp_b==0)break;
     T2_ChartFX->OpenDataEx(COD_VALUES,1,T[1].Grow_Out_days+T[1].Days_Of_Culture);

      for (i=0;i<=T[1].Grow_Out_days+T[1].Days_Of_Culture-1;i++)
       {
         T2_ChartFX->ThisSerie= 0;
         T2_ChartFX->Value[i]= T[1].sp_fin_wt * pow((1 - exp(-T[1].sp_gr_fac * i/365
)),T[1].sp_b);//{sets the Y value}
         T2_ChartFX->XValue[i]= i;     // {sets the X value}
       }
     T2_ChartFX->CloseData(COD_VALUES);
     T2_W_C->Text= FloatToStr(T[1].sp_fin_wt * pow((1 - exp(-T[1].sp_gr_fac *
(float)(T[1].Days_Of_Culture)/365 )),T[1].sp_b));
      break;

   case 2:
     if(T[2].sp_fin_wt==0 || T[2].sp_gr_fac==0 || T[2].sp_b==0)break;
```

```
    T3_ChartFX->OpenDataEx(COD_VALUES,1,T[2].Grow_Out_days+T[2].Days_Of_Culture);

     for (i=0;i<=T[2].Grow_Out_days+T[2].Days_Of_Culture-1;i++)
      {
        T3_ChartFX->ThisSerie= 0;
        T3_ChartFX->Value[i]= T[2].sp_fin_wt * pow((1 - exp(-T[2].sp_gr_fac * i/365
)),T[2].sp_b);//{sets the Y value}
        T3_ChartFX->XValue[i]= i;     // {sets the X value}
      }
     T3_ChartFX->CloseData(COD_VALUES);
     T3_W_C->Text= FloatToStr(T[2].sp_fin_wt * pow((1 - exp(-T[2].sp_gr_fac *
(float)(T[2].Days_Of_Culture)/365 )),T[2].sp_b));
     break;

   case 3:
     if(T[3].sp_fin_wt==0 || T[3].sp_gr_fac==0 || T[3].sp_b==0)break;
     T4_ChartFX->OpenDataEx(COD_VALUES,1,T[j].Grow_Out_days+T[j].Days_Of_Culture);

     for (i=0;i<=T[3].Grow_Out_days+T[3].Days_Of_Culture-1;i++)
      {
        T4_ChartFX->ThisScric= 0;
        T4_ChartFX->Value[i]= T[3].sp_fin_wt * pow((1 - exp(-T[3].sp_gr_fac * i/365
)),T[3].sp_b);//{sets the Y value}
        T4_ChartFX->XValue[i]= i;     // {sets the X value}
      }
     T4_ChartFX->CloseData(COD_VALUES);
     T4_W_C->Text= FloatToStr(T[3].sp_fin_wt * pow((1 - exp(-T[3].sp_gr_fac *
(float)(T[3].Days_Of_Culture)/365 )),T[3].sp_b));
     break;
   }
 }



}
//---------------------------------------------------------------------------

void __fastcall T_growth::T3_l_TChange(TObject *Sender)
{
 T[Tanks->PageIndex].Grow_Out_days=atoi(T_growth::T3_l_T->Text.c_str());
}
//---------------------------------------------------------------------------
```

_feed.cpp

```
// _feed.cpp
// 4/1/98
//need to total feed consumption
//need to log the calibration values
//need to put the exception routine in
//need to read the calibration values on start up
//need to chart the feed input from the log
```

```
//tie dumping of waste to feeding
//
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "_feed.h"
#include   <dos.h>
#include  <math.h>
#include "err_F.h"

//---------------------------------------------------------------------------
#pragma resource "*.dfm"

int change_to_mil_time(int);

extern struct time t;
extern struct date dt;
extern int day;
bool PRIME=false,FEED=false,done=false;

TFeed_sch *Feed_sch;
char PRMN;
float start_t,cal_feed_time,cal_feed_amt,AF1,AF2,AF3,AF4;
float raw_fced_t;
double fraction, integer;
int tot_P_time,whole;  //total prime time
int PT1,PT2,PT3,PT4;  //prime time intervals
int tot_F_time,FDN;
extern unsigned char agua_add;
extern unsigned char num_tanks;
struct feed_status{
  unsigned char agua_add;   //address of AGUA feed
  int                       AF_1;       //feeding time of feeder 1
  int                       AF_2;       //feeding time of feeder 2
  int                       AF_3;       //feeding time of feeder 3
  int                       AF_4;       //feeding time of feeder 4
  int                       AFA_1;            //time for feeder 1 to turn on (1/4 sec 0-999)
  int                       AFA_2;            //time for feeder 2 to turn on (1/4 sec 0-999)
  int                       AFA_3;            //time for feeder 3 to turn on (1/4 sec 0-999)
  int                       AFA_4;            //time for feeder 4 to turn on (1/4 sec 0-999)
  unsigned char STATUS;     //status flags for feeding done
  int                       t_s;              //second of feeding update
  int                       t_m;              //minute of feeding update
  int                       t_h;              //hour of feeding update
  int                       d_d;       //day of feeding
  int               d_m;       //month of feeding
}AF;

struct feed_log
{
  unsigned char agua_add; //address of the feeder unit
  float                     AF_cal1;          //feed delivered grams/sec
  float                     AF_cal2;          //feed delivered grams/sec
```

```
  float                    AF_cal3;          //feed delivered grams/sec
  float                    AF_cal4;          //feed delivered grams/sec
  float                    AF_Amt_D1;  //amount feed on a date
  float                    AF_Amt_D2;  //amount feed on a date
  float                    AF_Amt_D3;  //amount feed on a date
  float                    AF_Amt_D4;  //amount feed on a date
  float                    Tot_Feed_D1;//total feed delivered
  float                    Tot_Feed_D2;//total feed delivered
  float                    Tot_Feed_D3;//total feed delivered
  float                    Tot_Feed_D4;//total feed delivered
  int                      t_s;              //second of feeding
  int                      t_m;              //minute of feeding
  int                      t_h;              //hour of feeding
  int                      d_d;        //day of feeding
  int              d_m;     //month of feeding
  int                      d_y;              //year of feeding
  bool                     A_M_1;    //feeding mode true=automatic  false=manual
  bool                     A_M_2;    //feeding mode true=automatic  false=manual
  bool                     A_M_3;    //feeding mode true=automatic  false=manual
  bool                     A_M_4;    //feeding mode true=automatic  false=manual
}FL;

extern struct growth_data
{
   int  tank_num;       //tank number
   int  num_fish;       //number of fish in tank
         float sp_gr_fac;       //growth factor K
   float sp_b;          //b factor
   float sp_fin_wt;     //final expected wt
   float begin_wt;      //begining wt
   float present_wt;    //present weight
   float market_weight;   //market weight of fish
   float feed_rate;     //% body wieght to feed at
   int  Start_Day_age;      //days old when culture began
   int  Start_Date;     //day culture began (0-365)
   int  Grow_Out_days;  //number of days until harvest size
   int  Days_Of_Culture;  //number of days the culture
   char  *sp_name;
}T[4];

  void Log_Feed_Stat(void);
  void Log_Feed_Data(void);
  void  Adjust_feed_level(void);
//---------------------------------------------------------------------------
__fastcall TFeed_sch::TFeed_sch(TComponent* Owner)
        : TForm(Owner)
{
 AF.agua_add=agua_add;
 FL.agua_add=agua_add;
 AF_add->Text=(TFeed_sch::AF_add->Text.c_str());
 agua_add=atoi(TFeed_sch::AF_add->Text.c_str());

}
```

```
//---------------------------------------------------------------------------

void __fastcall TFeed_sch::FormActivate(TObject *Sender)
{
  FILE *f,*j,*g;

  f= fopen ("AGUAFEED.TXT", "r");
  if (f==NULL)return;

  fscanf(f,"%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",
                &AF.agua_add,
       &AF.AF_1,&AF.AF_2,&AF.AF_3,&AF.AF_4,
       &AF.AFA_1,&AF.AFA_2,&AF.AFA_3,&AF.AFA_4,
       &AF.STATUS,
       &AF.t_s,&AF.t_m,&AF.t_h,&AF.d_d,&AF.d_m);
  fclose(f);

        g= fopen ("AGUAFEED_LAST.TXT", "r");
  if (g==NULL)return;

  fscanf(g,"%d %f %f %f %f %f %f %f %f %f %d %d %d %d %d %d %d %d",

        &FL.agua_add,&FL.AF_cal1,&FL.AF_cal2,&FL.AF_cal3,&FL.AF_cal4,&FL.AF_Amt_D1,
       &FL.AF_Amt_D2,&FL.AF_Amt_D3,&FL.AF_Amt_D4,&FL.Tot_Feed_D1,&FL.Tot_Feed_D2,
       &FL.Tot_Feed_D3,&FL.Tot_Feed_D4,&FL.t_s,&FL.t_m,&FL.t_h,&FL.d_d,&AF.d_m,
       &FL.d_y,&FL.A_M_1,&FL.A_M_2,&FL.A_M_3,&FL.A_M_4);
  fclose(g);

   int k;

 for(k=0;k<=3;k++)
 {
  switch (k)
  {
   case 0:
   j= fopen ("GROWTH[0].TXT", "r");
   if (j==NULL)return;
   break;
   case 1:
   j= fopen ("GROWTH[1].TXT", "r");
   if (j==NULL)return;
   break;
   case 2:
   j= fopen ("GROWTH[2].TXT", "r");
   if (j==NULL)return;
   break;
   case 3:
   j= fopen ("GROWTH[3].TXT", "r");
   if (j==NULL)return;
   break;
  }

  fscanf(j,"%d %f %f %f %f %f %d %d %d %d %f %d %c",
```

```
                &T[k].tank_num,&T[k].sp_gr_fac,&T[k].sp_b,&T[k].sp_fin_wt,
    &T[k].begin_wt,&T[k].present_wt,&T[k].feed_rate,
    &T[k].Start_Day_age,&T[k].Start_Date,&T[k].Grow_Out_days,
    &T[k].Days_Of_Culture,&T[k].market_weight,&T[k].num_fish,
    &T[k].sp_name);
fclose(j);
}

AF_add->Text=IntToStr(AF.agua_add);
M_F_T1->Text=IntToStr(change_to_mil_time(AF.AF_1));
M_F_T2->Text=IntToStr(change_to_mil_time(AF.AF_2));
M_F_T3->Text=IntToStr(change_to_mil_time(AF.AF_3));
M_F_T4->Text=IntToStr(change_to_mil_time(AF.AF_4));
F_I1->Text=FloatToStr(((float)AF.AFA_1)/4.0);
Label26->Caption=FloatToStr(fraction);
F_I2->Text=FloatToStr((float)AF.AFA_2/4.0);
F_I3->Text=FloatToStr((float)AF.AFA_3/4.0);
F_I4->Text=FloatToStr((float)AF.AFA_4/4.0);
T1BW->Text=FloatToStr(T[0].feed_rate*100);
T2BW->Text=FloatToStr(T[1].feed_rate*100);
T3BW->Text=FloatToStr(T[2].feed_rate*100);
T4BW->Text=FloatToStr(T[3].feed_rate*100);
T1_NF->Text=IntToStr(T[0].num_fish);
T2_NF->Text=IntToStr(T[1].num_fish);
T3_NF->Text=IntToStr(T[2].num_fish);
T4_NF->Text=IntToStr(T[3].num_fish);
Adjust_feed_level();
Edit9->Text=FloatToStr(FL.AF_cal1);
Edit10->Text=FloatToStr(AF1/FL.AF_cal1);
Edit8->Text=FloatToStr(fraction);
Edit7->Text=IntToStr((float)whole/.25);
F_I1->Text=FloatToStr((float)AF.AFA_1/4.0);
      F_I2->Text=FloatToStr((float)AF.AFA_2/4.0);
      F_I3->Text=FloatToStr((float)AF.AFA_3/4.0);
      F_I4->Text=FloatToStr((float)AF.AFA_4/4.0);
T1_FW->Text=FloatToStr(T[0].present_wt);
T2_FW->Text=FloatToStr(T[1].present_wt);
T3_FW->Text=FloatToStr(T[2].present_wt);
T4_FW->Text=FloatToStr(T[3].present_wt);
T1_AF->Text=FloatToStr(AF1);
T2_AF->Text=FloatToStr(AF2);
T3_AF->Text=FloatToStr(AF3);
T4_AF->Text=FloatToStr(AF4);
cal_1->Text=FloatToStr(FL.AF_cal1);
cal_2->Text=FloatToStr(FL.AF_cal2);
cal_3->Text=FloatToStr(FL.AF_cal3);
cal_4->Text=FloatToStr(FL.AF_cal4);
if(FL.A_M_1)A_Mode1->Caption="A";
else A_Mode1->Caption="M";
if(FL.A_M_2)A_Mode2->Caption="A";
else A_Mode2->Caption="M";
if(FL.A_M_3)A_Mode3->Caption="A";
else A_Mode3->Caption="M";
if(FL.A_M_4)A_Mode4->Caption="A";
```

```
  else A_Mode4->Caption="M";
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::AF_addChange(TObject *Sender)
{

  agua_add=atoi(TFeed_sch::AF_add->Text.c_str());

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::P_F1MouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
   PRMN=1;  //prime the number 1 feeder

   gettime(&t);
   start_t=t.ti_sec + t.ti_hund*.01;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::P_F1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{    //prime from screen by holding down the prime button 1/4 sec interval min
   gettime(&t);
   if(PT11>0)tot_P_time=PT11;
   else
   tot_P_time=((int)ceil((t.ti_sec + t.ti_hund*.01)-start_t))*4;
   PT1->Text=IntToStr(tot_P_time/4);
   PRIME=true;
}

//---------------------------------------------------------------------------
void __fastcall TFeed_sch::P_F2MouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{

   PRMN=2;    //prime the number 2 feeder
   gettime(&t);
   start_t=t.ti_sec + t.ti_hund*.01;

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::P_F2MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{      //prime from screen by holding down the prime button 1/4 sec interval min
   gettime(&t);
   if(PT12>0)tot_P_time=PT12;
   else
   tot_P_time=((int)ceil((t.ti_sec + t.ti_hund*.01)-start_t))*4;
   PT2->Text=IntToStr(tot_P_time/4);
   PRIME=true;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::P_F3MouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
```

```
{
        PRMN=3;    //prime the number 3 feeder
  gettime(&t);
  start_t=t.ti_scc + t.ti_hund*.01;

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::P_F3MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{     //prime from screen by holding down the prime button 1/4 scc interval min
  gettime(&t);
  if(PT13>0)tot_P_time=PT13;
  else
  tot_P_time=((int)ceil((t.ti_scc + t.ti_hund*.01)-start_t))*4;
  PT3->Text=IntToStr(tot_P_time/4);
  PRIME=true;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::P_F4MouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
        PRMN=4;    //prime the number 4 feeder
  gettime(&t);
  start_t=t.ti_scc + t.ti_hund*.01;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::P_F4MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{     //prime from screen by holding down the prime button 1/4 sec interval min
      gettime(&t);
  if(PT14>0)tot_P_time=PT14;
  else
  tot_P_time=((int)ceil((t.ti_scc + t.ti_hund*.01)-start_t))*4;
  PT4->Text=IntToStr(tot_P_time/4);
  PRIME=true;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::M_F_T1Change(TObject *Sender)
{           //sets time to feed for feedr 1
        int AFTH,AFt,AFTM;
  int tot_min;

  AFt=atoi(TFeed_sch::M_F_T1->Text.c_str());

  AFTH=AFt/100;          //gives hr
  AFTM=AFt-(AFTH*100);  //gives min
  tot_min=AFTM+(AFTH*60);  //gives total min of the time
  AF.AF_1=tot_min;       //time to feed AGUA feeder 2

}
//---------------------------------------------------------------------------

void __fastcall TFeed_sch::M_F_T1Exit(TObject *Sender)
{     //makes the time to feed into an integer
```

```
    int AFTH,AFt,AFTM;
    int tot_min,T1_h,T1_m;

    AFt=atoi(TFeed_sch::M_F_T1->Text.c_str());

    AFTH=AFt/100;        //gives hr
    AFTM=AFt-(AFTH*100);  //gives min
    tot_min=AFTM+(AFTH*60);   //gives total min of the time
    AF.AF_1=tot_min;        //time to feed AGUA feeder 1
    while(AF.AF_1==AF.AF_3 || AF.AF_1==AF.AF_2 || AF.AF_1==AF.AF_4)
          {       AF.AF_1++;
                  T1_h=AF.AF_1/60;  //hours
                  T1_m=AF.AF_1-(T1_h*60);
                  M_F_T1->Text=IntToStr((T1_h*100)+T1_m);
     }
    M_F_T1->Color=clYellow;

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::M_F_T2Change(TObject *Sender)
{           //sets time to feed for feedr 2
          int AFTH,AFt,AFTM;
    int tot_min;

    AFt=atoi(TFeed_sch::M_F_T2->Text.c_str());

    AFTH=AFt/100;        //gives hr
    AFTM=AFt-(AFTH*100);  //gives min
    tot_min=AFTM+(AFTH*60);   //gives total min of the time
    AF.AF_2=tot_min;        //time to feed AGUA feeder 2

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::M_F_T2Exit(TObject *Sender)
{
          int AFTH,AFt,AFTM;
    int tot_min,T2_h,T2_m;

    AFt=atoi(TFeed_sch::M_F_T2->Text.c_str());

    AFTH=AFt/100;        //gives hr
    AFTM=AFt-(AFTH*100);  //gives min
    tot_min=AFTM+(AFTH*60);   //gives total min of the time
    AF.AF_2=tot_min;        //time to feed AGUA feeder 1
    while(AF.AF_2==AF.AF_3 || AF.AF_2==AF.AF_4 || AF.AF_2==AF.AF_1)
    {
     AF.AF_2++;
     T2_h=AF.AF_2/60;  //hours
     T2_m=AF.AF_2-(T2_h*60);
     M_F_T2->Text=IntToStr((T2_h*100)+T2_m);
    }
    M_F_T2->Color=clYellow;

}
```

```
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::M_F_T3Change(TObject *Sender)
{                   //sets time to feed for feedr 3
          int AFTH,AFt,AFTM;
   int tot_min;

   AFt=atoi(TFeed_sch::M_F_T3->Text.c_str());

   AFTH=AFt/100;          //gives hr
   AFTM=AFt-(AFTH*100);  //gives min
   tot_min=AFTM+(AFTH*60);   //gives total min of the time
   AF.AF_3=tot_min;          //time to feed AGUA feeder 3
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::M_F_T3Exit(TObject *Sender)
{
   int AFTH,AFt,AFTM;
   int tot_min,T3_h,T3_m;

   AFt=atoi(TFeed_sch::M_F_T3->Text.c_str());

   AFTH=AFt/100;          //gives hr
   AFTM=AFt-(AFTH*100);  //gives min
   tot_min=AFTM+(AFTH*60);   //gives total min of the time
   AF.AF_3=tot_min;          //time to feed AGUA feeder 1
   while(AF.AF_3==AF.AF_4 || AF.AF_3==AF.AF_2 || AF.AF_3==AF.AF_1)
   {
    AF.AF_3++;
    T3_h=AF.AF_3/60;  //hours
    T3_m=AF.AF_3-(T3_h*60);  //minutes
    M_F_T3->Text=IntToStr((T3_h*100)+T3_m);
   }
   M_F_T3->Color=clYellow;

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::M_F_T4Change(TObject *Sender)
{                   //sets time to feed for feedr 4
          int AFTH,AFt,AFTM;
   int tot_min;

   AFt=atoi(TFeed_sch::M_F_T4->Text.c_str());

   AFTH=AFt/100;          //gives hr
   AFTM=AFt-(AFTH*100);  //gives min
   tot_min=AFTM+(AFTH*60);   //gives total min of the time
   AF.AF_4=tot_min;          //time to feed AGUA feeder 4
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::M_F_T4Exit(TObject *Sender)
{
   int AFTH,AFt,AFTM;
   int tot_min,T4_h,T4_m;
```

```
AFt=atoi(TFeed_sch::M_F_T4->Text.c_str());

AFTH=AFt/100;        //gives hr
AFTM=AFt-(AFTH*100);  //gives min
tot_min=AFTM+(AFTH*60);   //gives total min of the time
AF.AF_4=tot_min;        //time to feed AGUA feeder 1
while(AF.AF_4==AF.AF_3 || AF.AF_4==AF.AF_2 || AF.AF_4==AF.AF_1)
{
 AF.AF_4++;
 T4_h=AF.AF_4/60;  //hours
 T4_m=AF.AF_4-(T4_h*60);
 M_F_T4->Text=IntToStr((T4_h*100)+T4_m);
 }
 M_F_T4->Color=clYellow;

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::F_11Change(TObject *Sender)
{       //amount to feed feeder 1 (1/4 sec is the smallest interval)
  AF.AFA_1=(int)(4*atof(TFeed_sch::F_11->Text.c_str()));

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::F_12Change(TObject *Sender)
{     //amount to feed feeder 2 (1/4 sec is the smallest interval)
        AF.AFA_2=(int)(4*atof(TFeed_sch::F_12->Text.c_str()));

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::F_13Change(TObject *Sender)
{    //amount to feed feeder 3 (1/4 sec is the smallest interval)
        AF.AFA_3=(int)(4*atof(TFeed_sch::F_13->Text.c_str()));

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::F_14Change(TObject *Sender)
{     //amount to feed feeder 4 (1/4 sec is the smallest interval)
        AF.AFA_4=(int)(4*atof(TFeed_sch::F_14->Text.c_str()));

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::PT1Change(TObject *Sender)
{      //manual input of prime time (1/4 sec min and interval)
  PTI1=(int)(4*atof(TFeed_sch::PT1->Text.c_str()));
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::PT2Change(TObject *Sender)
{      //manual input of prime time (1/4 sec min and interval)
  PTI2=(int)(4*atof(TFeed_sch::PT2->Text.c_str()));
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::PT3Change(TObject *Sender)
{     //manual input of prime time (1/4 sec min and interval)
        PTI3=(int)(4*atof(TFeed_sch::PT3->Text.c_str()));
```

```
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::PT4Change(TObject *Sender)
{    //manual input of prime time (1/4 sec min and interval)
        PTI4=(int)(4*atof(TFeed_sch::PT4->Text.c_str()));
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::Timer1Timer(TObject *Sender)
{   //checks every minute to see if feeding required
        int CT_min; //current time in minutes

        gettime(&t);
            //reset feeder flags every midnight
   if(t.ti_hour==0 && t.ti_min<=1 && done==false)
   {
        AF.STATUS=0x0000;   //reset feed flag
     Adjust_feed_level();
     Log_Feed_Stat();
     done=true;          //reset feed flag has occured
        T1_FW->Text=FloatToStr(T[0].present_wt);
                        T2_FW->Text=FloatToStr(T[1].present_wt);
                        T3_FW->Text=FloatToStr(T[2].present_wt);
                        T4_FW->Text=FloatToStr(T[3].present_wt);
                        T1_AF->Text=FloatToStr(AF1);
                        T2_AF->Text=FloatToStr(AF2);
                        T3_AF->Text=FloatToStr(AF3);
                        T4_AF->Text=FloatToStr(AF4);
   }

   CT_min=(t.ti_hour*60)+t.ti_min;  //get current time in minutes

   if(CT_min>=AF.AF_1 && !(AF.STATUS & 0x08) && AF.AFA_1>0 && AF.AF_1>0)
        {                   //if the current time is => time of feeding
     FEED=true;          //and the feeding has not occurred
     FDN=1;              //and data entered (feeding cannot occur at midnight
     tot_F_time=AF.AFA_1;
     FL.AF_Amt_D1=(((float)AF.AFA_1/4)*FL.AF_cal1);  //amount fed at this feeding in grams
     FL.Tot_Feed_D1=FL.Tot_Feed_D1+(((float)AF.AFA_1/4)*FL.AF_cal1); //total feed delivered to
date
     Log_Feed_Data();
     }
   if(CT_min>=AF.AF_2 && !(AF.STATUS & 0x04) && AF.AFA_2>0 && AF.AF_2>0)
        {
     FEED=true;
     FDN=2;
     tot_F_time=AF.AFA_2;
     FL.AF_Amt_D2=(((float)AF.AFA_2/4)*FL.AF_cal2);  //amount fed at this feeding in grams
     FL.Tot_Feed_D2=FL.Tot_Feed_D2+(((float)AF.AFA_2/4)*FL.AF_cal2);
     Log_Feed_Data();
     }
   if(CT_min>=AF.AF_3 && !(AF.STATUS & 0x02) && AF.AFA_3>0 && AF.AF_3>0)
        {
     FEED=true;
     FDN=3;
```

```
      tot_F_time=AF.AFA_3;
      FL.AF_Amt_D3=(((float)AF.AFA_3/4)*FL.AF_cal3);  //amount fed at this feeding in grams
      FL.Tot_Feed_D3=FL.Tot_Feed_D3+(((float)AF.AFA_3/4)*FL.AF_cal3);
      Log_Feed_Data();
     }
   if(CT_min>=AF.AF_4 && !(AF.STATUS & 0x01) && AF.AFA_4>0 && AF.AF_4>0)
      {
      FEED=true;
      FDN=4;
      tot_F_time=AF.AFA_4;
      FL.AF_Amt_D4=(((float)AF.AFA_4/4)*FL.AF_cal4);  //amount fed at this feeding in grams
      FL.Tot_Feed_D4=FL.Tot_Feed_D4+(((float)AF.AFA_4/4)*FL.AF_cal4);
      Log_Feed_Data();
     }
   if(AF.STATUS==15)done=false;   //all feeding has occured reset feeding status
}

//---------------------------------------------------------------------------
void __fastcall TFeed_sch::Log_SchClick(TObject *Sender)
{
  Log_Feed_Stat();  //manually stores the feed data
}
//---------------------------------------------------------------------------

int change_to_mil_time(int time_min)
{
  int T_h,T_m;

  T_h=time_min/60;
  T_m=time_min-(T_h*60);
  return((T_h*100)+T_m);

}

//---------------------------------------------------------------------------
void Log_Feed_Stat(void)
{    //manually store the feed data
        FILE *f,*j;
   f= fopen ("AGUAFEED.TXT", "w");
   if (f==NULL)return;



  gettime(&t);
  getdate(&dt);
  AF.t_s = t.ti_sec;
  AF.t_m = t.ti_min;
  AF.t_h = t.ti_hour;
  AF.d_d = dt.da_day;
  AF.d_m = dt.da_mon;

  fprintf(f,"%d %d %d %d %d %d %d %d %d %d %d %d %d %d",
                AF.agua_add,
       AF.AF_1,AF.AF_2,AF.AF_3,AF.AF_4,
```

```
        AF.AFA_1,AF.AFA_2,AF.AFA_3,AF.AFA_4,
        AF.STATUS,
        AF.t_s,AF.t_m,AF.t_h,AF.d_d,AF.d_m);

  fprintf(f, "\n");
  fclose(f);

  int k;
  for(k=0;k<=3;k++)
  {
          switch (k)
          {
                  case 0:
          j= fopen ("GROWTH[0].TXT", "w");
          if (j==NULL)return;
          break;
          case 1:
          j= fopen ("GROWTH[1].TXT", "w");
          if (j==NULL)return;
      break;
          case 2:
          j= fopen ("GROWTH[2].TXT", "w");
          if (j==NULL)return;
          break;
          case 3:
          j= fopen ("GROWTH[3].TXT", "w");
          if (j==NULL)return;
          break;
  }

     fprintf(j,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t%d\t%d\t%d\t%4.2f\t%d\t%c\t",
                  T[k].tank_num,T[k].sp_gr_fac,T[k].sp_b,
        T[k].sp_fin_wt,T[k].begin_wt,T[k].present_wt,
        T[k].feed_rate,T[k].Start_Day_age,T[k].Start_Date,T[k].Grow_Out_days,
        T[k].Days_Of_Culture,T[k].market_weight,
        T[k].num_fish,T[k].sp_name);
    fclose(j);
  }

}
//---------------------------------------------------------------------------
void Log_Feed_Data()
{
          FILE * f;       //appends to the feeding log
    f= fopen ("AGUAFEED_LOG.TXT", "at");
    if (f==NULL)return;

    gettime(&t);
    getdate(&dt);
    FL.t_s = t.ti_sec;
    FL.t_m = t.ti_min;
    FL.t_h = t.ti_hour;
    FL.d_d = dt.da_day;
    FL.d_m = dt.da_mon;
```

```
  FL.d_y = dt.da_year;


fprintf(f,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\
t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t",
                FL.agua_add.FL.AF_cal1,FL.AF_cal2,FL.AF_cal3,FL.AF_cal4.FL.AF_Amt_D1.
        FL.AF_Amt_D2.FL.AF_Amt_D3.FL.AF_Amt_D4.FL.Tot_Fced_D1.FL.Tot_Fced_D2.
        FL.Tot_Fced_D3.FL.Tot_Fced_D4.FL.t_s.FL.t_m.FL.t_h.FL.d_d.FL.d_m.
        FL.d_y.FL.A_M_1.FL.A_M_2.FL.A_M_3.FL.A_M_4);
  fprintf(f, "\n");
  fclose(f);

FILE * g;    //writes to the fced file that keeps track of fced delivered and
                //settings for fced delivery
  g= fopen ("AGUAFEED_LAST.TXT", "w");
  if (g==NULL)return;

  fprintf(g,"%d %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %d %d
%d %d %d %d %d %d %d %d %d %d".
                FL.agua_add.FL.AF_cal1,FL.AF_cal2,FL.AF_cal3,FL.AF_cal4.FL.AF_Amt_D1.
        FL.AF_Amt_D2.FL.AF_Amt_D3.FL.AF_Amt_D4.FL.Tot_Fced_D1.FL.Tot_Fced_D2.
        FL.Tot_Fced_D3.FL.Tot_Fced_D4.FL.t_s.FL.t_m.FL.t_h.FL.d_d.AF.d_m.
        FL.d_y.FL.A_M_1.FL.A_M_2.FL.A_M_3.FL.A_M_4);
  fclose(g);


//---------------------------------------------------------------------------
}
void __fastcall TFced_sch::FN_ChangeClick(TObject *Sender, TUDBtnType Button)
{
feed_num->Text=FN_Change->Position;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::cal_feederMouseDown(TObject *Sender,
        TMouseButton Button, TShiftState Shift, int X, int Y)
{

        gettime(&t);
  start_t=t.ti_sec + t.ti_hund*.01;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::cal_feederMouseUp(TObject *Sender,
        TMouseButton Button, TShiftState Shift, int X, int Y)
{
        float cal_feed,t_sf;
  int t_s;

        gettime(&t);
  cal_feed=(t.ti_sec + t.ti_hund*.01)-start_t;
  if(cal_feed<=0)cal_feed=.25;
  t_s=(int)cal_feed;      //whole integer seconds
  t_sf=((float)((int)((cal_feed-t_s)*100)/25))*.25;     //fraction of seconds
  cal_feed=(float)t_s+t_sf;
//  tot_P_time=((int)ceil((t.ti_sec + t.ti_hund*.01)-start_t))*4;
```

```
        tot_P_time=(int)(cal_feed/.25);
    Label31->Caption=IntToStr(tot_P_time);
    Cal_time->Text=FloatToStr(cal_feed);

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::Cal_FeedClick(TObject *Sender)
{
  PRMN=atoi(TFeed_sch::feed_num->Text.c_str());     //prime the selected feeder
  PRIME=true;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::Cal_timeExit(TObject *Sender)
{
//manual input of calibrate time (1/4 sec min and interval)
   float cal_feed,t_sf;
   int t_s;
   cal_feed=atof(TFeed_sch::Cal_time->Text.c_str());
   t_s=(int)cal_feed;     //whole integer seconds
   t_sf=((float)((int)((cal_feed-t_s)*100)/25))*.25;     //fraction of seconds
   cal_feed=(float)t_s+t_sf;
   cal_feed_time=cal_feed;
        tot_P_time=(int)(cal_feed/.25);
   Label31->Caption=IntToStr(tot_P_time);
   Cal_time->Text=FloatToStr(cal_feed);
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::Set_CalibrationClick(TObject *Sender)
{
   float cal;

   if(cal_feed_amt<=0 || cal_feed_time<=0)return;//exception
        Label32->Visible=true;
   FN->Visible=true;
   Label33->Visible=true;
   CAL_SET->Visible=true;
   Label34->Visible=true;
   Timer2->Enabled=true;
   PRMN=atoi(TFeed_sch::feed_num->Text.c_str());
   cal_feed_time=atof(TFeed_sch::Cal_time->Text.c_str());
   cal_feed_amt=atof(TFeed_sch::AMT_D->Text.c_str());
   FN->Caption=IntToStr(PRMN);

   switch(PRMN)
   {
        case 1:{FL.AF_cal1=cal=cal_feed_amt/cal_feed_time;break;}
     case 2:{FL.AF_cal2=cal=cal_feed_amt/cal_feed_time;break;}
     case 3:{FL.AF_cal3=cal=cal_feed_amt/cal_feed_time;break;}
     case 4:{FL.AF_cal4=cal=cal_feed_amt/cal_feed_time;break;}
   }
   CAL_SET->Caption=FloatToStr(cal);
 Log_Feed_Data(); //stores the calibration change
}
//---------------------------------------------------------------------------
```

```
void __fastcall TFeed_sch::Timer2Timer(TObject *Sender)
{  //removes info after interval seconds
        Label32->Visible=false;
   FN->Visible=false;
   Label33->Visible=false;
   CAL_SET->Visible=false;
   Label34->Visible=false;
   Timer2->Enabled=false;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::AMT_DExit(TObject *Sender)
{
 cal_feed_amt=atof(TFeed_sch::AMT_D->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::SpeedButton2Click(TObject *Sender)
{
        AF.STATUS=0;   //resets the feed status bits
   Log_Feed_Stat();
}
//---------------------------------------------------------------------------
void  Adjust_feed_level(void)
{
        //checks the predicted growth of each tank and then resets the value for
   //the feeder for that tank
 int j,wt_to_time,part;


   if(FL.A_M_1)  //if the mode is set for auto then calculate weight
   {
          T[0].Days_Of_Culture=T[0].Start_Day_age+(day-T[0].Start_Date);
          T[0].present_wt=(T[0].sp_fin_wt*pow((1-exp(-
T[0].sp_gr_fac*(float)(T[0].Days_Of_Culture)/365)),T[0].sp_b));
   }
   if(FL.A_M_2)
   {
          T[1].Days_Of_Culture=T[1].Start_Day_age+(day-T[1].Start_Date);
          T[1].present_wt=(T[1].sp_fin_wt*pow((1-exp(-
T[1].sp_gr_fac*(float)(T[1].Days_Of_Culture)/365)),T[1].sp_b));
   }
   if(FL.A_M_3)
   {
          T[2].Days_Of_Culture=T[2].Start_Day_age+(day-T[2].Start_Date);
          T[2].present_wt=(T[2].sp_fin_wt*pow((1-exp(-
T[2].sp_gr_fac*(float)(T[2].Days_Of_Culture)/365)),T[2].sp_b));
   }
   if(FL.A_M_4)
   {
          T[3].Days_Of_Culture=T[3].Start_Day_age+(day-T[3].Start_Date);
          T[3].present_wt=(T[3].sp_fin_wt*pow((1-exp(-
T[3].sp_gr_fac*(float)(T[3].Days_Of_Culture)/365)),T[3].sp_b));
   }
```

```
AF1=(T[0].present_wt*T[0].num_fish*T[0].feed_rate);
AF2=(T[1].present_wt*T[1].num_fish*T[1].feed_rate);
AF3=(T[2].present_wt*T[2].num_fish*T[2].feed_rate);
AF4=(T[3].present_wt*T[3].num_fish*T[3].feed_rate);

raw_feed_t=AF1/FL.AF_cal1;
whole=floor(raw_feed_t);
fraction=modf(raw_feed_t,&integer);
if(fraction>0 && fraction<=.25)part=1;
if(fraction>.25 && fraction<=.50)part=2;
if(fraction>.5 && fraction<=.75)part=3;
if(fraction>.75 && fraction<=.99)part=4;
AF.AFA_1=(int)((float)floor(raw_feed_t)/.25)+part;


raw_feed_t=AF2/FL.AF_cal2;
whole=floor(raw_feed_t);
fraction=modf(raw_feed_t,&integer);
if(fraction>0 && fraction<=.25)part=1;
if(fraction>.25 && fraction<=.50)part=2;
if(fraction>.5 && fraction<=.75)part=3;
if(fraction>.75 && fraction<=.99)part=4;
AF.AFA_2=(int)((float)floor(raw_feed_t)/.25)+part;


raw_feed_t=AF3/FL.AF_cal3;
whole=floor(raw_feed_t);
fraction=modf(raw_feed_t,&integer);
if(fraction>0 && fraction<=.25)part=1;
if(fraction>.25 && fraction<=.50)part=2;
if(fraction>.5 && fraction<=.75)part=3;
if(fraction>.75 && fraction<=.99)part=4;
AF.AFA_3=(int)((float)floor(raw_feed_t)/.25)+part;


raw_feed_t=AF4/FL.AF_cal4;
whole=floor(raw_feed_t);
fraction=modf(raw_feed_t,&integer);
if(fraction>0 && fraction<=.25)part=1;
if(fraction>.25 && fraction<=.50)part=2;
if(fraction>.5 && fraction<=.75)part=3;
if(fraction>.75 && fraction<=.99)part=4;
AF.AFA_4=(int)((float)floor(raw_feed_t)/.25)+part;


        FILE *f;
 f= fopen ("AGUAFEED.TXT", "w");
 if (f==NULL)return;



gettime(&t);
getdate(&dt);
AF.t_s = t.ti_sec;
```

```
  AF.t_m = t.ti_min;
  AF.t_h = t.ti_hour;
  AF.d_d = dt.da_day;
  AF.d_m = dt.da_mon;

  fprintf(f,"%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",
                AF.agua_add,
        AF.AF_1,AF.AF_2,AF.AF_3,AF.AF_4,
        AF.AFA_1,AF.AFA_2,AF.AFA_3,AF.AFA_4,
        AF.STATUS,
        AF.t_s,AF.t_m,AF.t_h,AF.d_d,AF.d_m);

  fprintf(f, "\n");
  fclose(f);



}
//---------------------------------------------------------------------------

void __fastcall TFeed_sch::T1BWChange(TObject *Sender)
{
        T[0].feed_rate=.01*atof(TFeed_sch::T1BW->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::T2BWChange(TObject *Sender)
{
        T[1].feed_rate=.01*atof(TFeed_sch::T2BW->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::T3BWChange(TObject *Sender)
{
        T[2].feed_rate=.01*atof(TFeed_sch::T3BW->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::T4BWChange(TObject *Sender)
{
        T[3].feed_rate=.01*atof(TFeed_sch::T4BW->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::Button1Click(TObject *Sender)
{
  Adjust_feed_level();
  Edit1->Text=FloatToStr(T[0].present_wt);
  Edit7->Text=FloatToStr(T[0].num_fish);
  Edit8->Text=FloatToStr(T[0].feed_rate);
  Edit9->Text=FloatToStr(T[3].present_wt);
  Edit10->Text=IntToStr(T[0].Days_Of_Culture);
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::T1_NFChange(TObject *Sender)
{  //sets the number of fish in the tank and calcs how much to feed

        float feed_amount;
```

```
 T[0].num_fish=atoi(TFeed_sch::T1_NF->Text.c_str());

        feed_amount=T[0].num_fish*T[0].present_wt*T[0].feed_rate;
  T1_AF->Text=feed_amount;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::T2_NFChange(TObject *Sender)
{
 //sets the number of fish in the tank and calcs how much to feed
        float feed_amount;
  T[1].num_fish=atoi(TFeed_sch::T2_NF->Text.c_str());

        feed_amount=T[1].num_fish*T[1].present_wt*T[1].feed_rate;
  T2_AF->Text=feed_amount;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::T3_NFChange(TObject *Sender)
{

 //sets the number of fish in the tank and calcs how much to feed
        float feed_amount;
  T[2].num_fish=atoi(TFeed_sch::T3_NF->Text.c_str());

        feed_amount=T[2].num_fish*T[2].present_wt*T[2].feed_rate;
  T3_AF->Text=feed_amount;

}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::T4_NFChange(TObject *Sender)
{

 //sets the number of fish in the tank and calcs how much to feed
        float feed_amount;
  T[3].num_fish=atoi(TFeed_sch::T4_NF->Text.c_str());

        feed_amount=T[3].num_fish*T[3].present_wt*T[3].feed_rate;
  T4_AF->Text=feed_amount;
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::TabSheet3Enter(TObject *Sender)
{
  FILE *j,*g;

        g= fopen ("AGUAFEED_LAST.TXT", "r");
  if (g==NULL)return;

  fscanf(g,"%d %f %f %f %f %f %f %f %f %f %f %d %d %d %d %d",

        &FL.agua_add,&FL.AF_cal1,&FL.AF_cal2,&FL.AF_cal3,&FL.AF_cal4,&FL.AF_Amt_D1,
       &FL.AF_Amt_D2,&FL.AF_Amt_D3,&FL.AF_Amt_D4,&FL.Tot_Feed_D1,&FL.Tot_Feed_D2,
       &FL.Tot_Feed_D3,&FL.Tot_Feed_D4,&FL.t_s,&FL.t_m,&FL.t_h,&FL.d_d,&AF.d_m,
       &FL.d_y);
 fclose(g);
```

```
  int k;

for(k=0;k<=3;k++)
{
 switch (k)
 {
 case 0:
 j= fopen ("GROWTH[0].TXT", "r");
 if (j==NULL)return;
 break;
 case 1:
 j= fopen ("GROWTH[1].TXT", "r");
 if (j==NULL)return;
 break;
 case 2:
 j= fopen ("GROWTH[2].TXT", "r");
 if (j==NULL)return;
 break;
 case 3:
 j= fopen ("GROWTH[3].TXT", "r");
 if (j==NULL)return;
 break;
 }

 fscanf(j,"%d %f %f %f %f %f %d %d %d %d %f %d %c",
                &T[k].tank_num,&T[k].sp_gr_fac,&T[k].sp_b,&T[k].sp_fin_wt,
       &T[k].begin_wt,&T[k].present_wt,&T[k].feed_rate,
       &T[k].Start_Day_age,&T[k].Start_Date,&T[k].Grow_Out_days,
       &T[k].Days_Of_Culture,&T[k].market_weight,&T[k].num_fish,
       &T[k].sp_name);
fclose(j);
}


AF_add->Text=IntToStr(AF.agua_add);
M_F_T1->Text=IntToStr(change_to_mil_time(AF.AF_1));
M_F_T2->Text=IntToStr(change_to_mil_time(AF.AF_2));
M_F_T3->Text=IntToStr(change_to_mil_time(AF.AF_3));
M_F_T4->Text=IntToStr(change_to_mil_time(AF.AF_4));
F_I1->Text=FloatToStr((float)AF.AFA_1/4.0);
F_I2->Text=FloatToStr((float)AF.AFA_2/4.0);
F_I3->Text=FloatToStr((float)AF.AFA_3/4.0);
F_I4->Text=FloatToStr((float)AF.AFA_4/4.0);
T1BW->Text=FloatToStr(T[0].feed_rate*100);
T2BW->Text=FloatToStr(T[1].feed_rate*100);
T3BW->Text=FloatToStr(T[2].feed_rate*100);
T4BW->Text=FloatToStr(T[3].feed_rate*100);
T1_NF->Text=IntToStr(T[0].num_fish);
T2_NF->Text=IntToStr(T[1].num_fish);
T3_NF->Text=IntToStr(T[2].num_fish);
T4_NF->Text=IntToStr(T[3].num_fish);
Adjust_feed_level();
Edit9->Text=FloatToStr(FL.AF_cal1);
Edit10->Text=FloatToStr(AF1/FL.AF_cal1);
```

```
  Edit8->Text=FloatToStr(fraction);
  Edit7->Text=IntToStr(AF.AFA_1);
  T1_FW->Text=FloatToStr(T[0].present_wt);
  T2_FW->Text=FloatToStr(T[1].present_wt);
  T3_FW->Text=FloatToStr(T[2].present_wt);
  T4_FW->Text=FloatToStr(T[3].present_wt);
  T1_AF->Text=FloatToStr(AF1);
  T2_AF->Text=FloatToStr(AF2);
  T3_AF->Text=FloatToStr(AF3);
  T4_AF->Text=FloatToStr(AF4);
}
//---------------------------------------------------------------------------

void __fastcall TFeed_sch::T1_FWChange(TObject *Sender)
{
 T[0].present_wt=atof(TFeed_sch::T1_FW->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::T2_FWChange(TObject *Sender)
{
 T[1].present_wt=atof(TFeed_sch::T2_FW->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::T3_FWChange(TObject *Sender)
{
 T[2].present_wt=atof(TFeed_sch::T3_FW->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::T4_FWChange(TObject *Sender)
{
 T[3].present_wt=atof(TFeed_sch::T4_FW->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::A_Mode1Click(TObject *Sender)
{
 if(FL.A_M_1){FL.A_M_1=false;A_Mode1->Caption="M";return;}
 if(!FL.A_M_1){FL.A_M_1=true;A_Mode1->Caption="A";return;}
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::A_Mode2Click(TObject *Sender)
{
 if(FL.A_M_2){FL.A_M_2=false;A_Mode2->Caption="M";return;}
 if(!FL.A_M_2){FL.A_M_2=true;A_Mode2->Caption="A";return;}
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::A_Mode3Click(TObject *Sender)
{
 if(FL.A_M_3){FL.A_M_3=false;A_Mode3->Caption="M";return;}
 if(!FL.A_M_3){FL.A_M_3=true;A_Mode3->Caption="A";return;}
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::A_Mode4Click(TObject *Sender)
{
 if(FL.A_M_4){FL.A_M_4=false;A_Mode4->Caption="M";return;}
```

```
 if(!FL.A_M_4){FL.A_M_4=true;A_Mode4->Caption="A";return;}
}
//---------------------------------------------------------------------------
void __fastcall TFeed_sch::FormClose(TObject *Sender, TCloseAction &Action)
{
 FILE * g;    //writes to the feed file that keeps track of feed delivered and
              //settings for feed delivery
  g= fopen ("AGUAFEED_LAST.TXT", "w");
  if (g==NULL)return;

  fprintf(g,"%d %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f %d %d
%d %d %d %d %d %d %d %d %d",
                FL.agua_add,FL.AF_cal1,FL.AF_cal2,FL.AF_cal3,FL.AF_cal4,FL.AF_Amt_D1,
       FL.AF_Amt_D2,FL.AF_Amt_D3,FL.AF_Amt_D4,FL.Tot_Feed_D1,FL.Tot_Feed_D2,
       FL.Tot_Feed_D3,FL.Tot_Feed_D4,FL.t_s,FL.t_m,FL.t_h,FL.d_d,AF.d_m,
       FL.d_y,FL.A_M_1,FL.A_M_2,FL.A_M_3,FL.A_M_4);
 fclose(g);
}
//---------------------------------------------------------------------------
```

oxygen.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "_oxygen.h"
#include <math.h>
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
ToxyF *oxyF;

float O2_level,water_temp,water_salinity,site_elevation;
extern float sp_O2_min,sp_O2_max,sp_temp_min,sp_temp_max,sp_pH_min,sp_pH_max;
extern float sp_alk_min,sp_alk_max,sp_CO2_min,sp_CO2_max,sp_NH4_min,sp_NH4_max;
extern float sp_NO2_min,sp_NO2_max,sp_hard_min,sp_hard_max,sp_chl_min,sp_chl_max;
extern float sp_sal_min,sp_sal_max;
extern String sp_name;
extern int Tank_Number;
extern struct growth_data
{
  int  tank_num;      //tank number
  int  num_fish;      //number of fish in tank
       float sp_gr_fac;      //growth factor
  float sp_b;         //b factor
  float sp_fin_wt;    //final expected wt after 1 year
  float begin_wt;     //begining wt
  float present_wt;   //present weight
  float market_weight;  //market weight of fish
  float feed_rate;    //% body wieght to feed at
  int  Start_Day;     //day culture was spawned
  int  Grow_Out_days;  //number of days until harvest size
  int  Days_Of_Culture;  //age of the culture
```

```
    char* sp_name;

};T[4];
//---------------------------------------------------------------------------
__fastcall ToxyF::ToxyF(TComponent* Owner)
        : TForm(Owner)
{

}
//---------------------------------------------------------------------------

float calc_Bp(float ht_meters)
{
        float Bp;

    Bp=pow(10,(2.880814-(ht_meters/19748.0))); // calculates barometric pressure
                                                                    // from sea
level (unadjusted)
    Bp=Bp*0.0013157895;  //converts to pressure in atmospheres
    return(Bp);

}

float calc_O2_level(float temp_C,float salinity,float elevation)
{
        float O2_level_max,T_K,chl,P_wv,theta,cor_O2_max;

    //converts to kelvin
    T_K=temp_C+273.15;
    //converts salinity to chlorinity
    chl=salinity/1.80655;
    //calculates maximum O2 level on temp and Salinity
    O2_level_max=exp((-139.34411+(1.575701e5/T_K)-(6.642308e7/(T_K*T_K))
                        +(1.2438e10/pow(T_K,3))
            +((8.621949e11/pow(T_K,4))*(-1.0)))
            -(( chl*( (3.1929e-2)-(1.9428e1/T_K)
            +(3.8673e3/(T_K*T_K)) ) )));
    //calculates partial pressure of water vapor
    P_wv=exp(11.8571-(3840.7/T_K)-(216961/(T_K*T_K)));
    //correction value
    theta=0.000975-(1.426e-5*temp_C)+(6.436e-8*temp_C*temp_C);
    cor_O2_max=O2_level_max*calc_Bp(elevation)
                        *( ( (1-(P_wv/calc_Bp(elevation)))
            *(1-(theta*calc_Bp(elevation)) )
            /( (1-P_wv)*(1-theta)) ) );

    return(cor_O2_max);
}


//---------------------------------------------------------------------------
void __fastcall ToxyF::calc_oxyClick(TObject *Sender)
{
        max_level->Text=FloatToStr(calc_O2_level(atof(ToxyF::temp->Text.c_str()),
```

```
   atof(ToxyF::sal->Text.c_str()),
   atof(ToxyF::el_M->Text.c_str()));
}
//---------------------------------------------------------------------------
void __fastcall ToxyF::FormActivate(TObject *Sender)
{


          sp_min->Text=FloatToStr(sp_O2_min);
   sp_max->Text=FloatToStr(sp_O2_max);
   // _spN->Caption=T[0].sp_name;

}
//---------------------------------------------------------------------------
```

system_design.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "_system_design.h"
#define PI    3.14159265359
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
Tsystem_design *system_design;
unsigned char   num_tanks;          //number of tanks in the system
float T_N=4,T_L=20,T_H=12,T_W=12;
float T_V=0;
float unit;
bool units=true;


//---------------------------------------------------------------------------
__fastcall Tsystem_design::Tsystem_design(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall Tsystem_design::Tank_TypeClick(TObject *Sender)
{


         if (Tank_Type->ItemIndex==0)

     {tank_len->Visible=false;t_l->Visible=false;}


   if (Tank_Type->ItemIndex==1)
       {tank_len->Visible=true;t_l->Visible=true;}

}
```

```
//---------------------------------------------------------------------------
void __fastcall Tsystem_design::tank_htChange(TObject *Sender)
{
        T_H=atof(Tsystem_design::tank_ht->Text.c_str());
   if (Tank_Type->ItemIndex==0)//tank is cylindrical
   if(T_N>0 && T_W>0 && T_H>0)calc_vol();
   if (Tank_Type->ItemIndex==1)//tank is rectangular
   if(T_N>0 && T_L>0 && T_W>0 && T_H>0)calc_vol();
}
//---------------------------------------------------------------------------
void __fastcall Tsystem_design::tank_wdChange(TObject *Sender)
{
        T_W=atof(Tsystem_design::tank_wd->Text.c_str());
   if (Tank_Type->ItemIndex==0)//tank is cylindrical
   if(T_N>0 && T_W>0 && T_H>0)calc_vol();
   if (Tank_Type->ItemIndex==1)//tank is rectangular
   if(T_N>0 && T_L>0 && T_W>0 && T_H>0)calc_vol();
}
//---------------------------------------------------------------------------
void __fastcall Tsystem_design::tank_lenChange(TObject *Sender)
{
        T_L=atof(Tsystem_design::tank_len->Text.c_str());
   if (Tank_Type->ItemIndex==1)//tank is rectangular
   if(T_N>0 && T_L>0 && T_W>0 && T_H>0)calc_vol();
}
//---------------------------------------------------------------------------
void __fastcall Tsystem_design::n_tChange(TObject *Sender)
{
        num_tanks=T_N=atoi(Tsystem_design::n_t->Text.c_str());
   if (Tank_Type->ItemIndex==0)//tank is cylindrical
   if(T_N>0 && T_W>0 && T_H>0)calc_vol();
   if (Tank_Type->ItemIndex==1)//tank is rectangular
   if(T_N>0 && T_L>0 && T_W>0 && T_H>0)calc_vol();
}
//---------------------------------------------------------------------------
void __fastcall Tsystem_design::UnitsClick(TObject *Sender)
{
        if (Units->ItemIndex==0)
        {
      units=true;
      Label3->Caption="Height of Tank (in)";
      Label4->Caption="Width of Tank (in)";
      t_l->Caption="Length of Tank (in)";
      Label5->Caption="gallons";

    }
  if (Units->ItemIndex==1)
                {
      units=false;
      Label3->Caption="Height of Tank (cm)";
      Label4->Caption="Width of Tank (cm)";
      t_l->Caption="Length of Tank (cm)";
      Label5->Caption="liters";
    }
```

```
}
//---------------------------------------------------------------------------
void __fastcall Tsystem_design::FormCreate(TObject *Sender)
{
          if(units)
          {
       Label3->Caption="Height of Tank (in)";
       Label4->Caption="Width of Tank (in)";
       t_l->Caption="Length of Tank (in)";

     }
   else
          {
       Label3->Caption="Height of Tank (cm)";
       Label4->Caption="Width of Tank (cm)";
       t_l->Caption="Length of Tank (cm)";
     }
   if (Tank_Type->ItemIndex==1)
        {tank_len->Visible=true;t_l->Visible=true;}
   if (Tank_Type->ItemIndex==0)//tank is cylindrical
   if(T_N>0 && T_W>0 && T_H>0)calc_vol();
   if (Tank_Type->ItemIndex==1)//tank is rectangular
   if(T_N>0 && T_L>0 && T_W>0 && T_H>0)calc_vol();
}
//---------------------------------------------------------------------------
void __fastcall Tsystem_design::calc_vol(void)
{
          if(units)unit=.004329;    //vol (gal)
   else unit= .001;              //vol (liter)

  if (Tank_Type->ItemIndex==1)
  {
  tank_vol->Text=FloatToStr(T_N*T_L*T_W*T_H*unit);
  T_V=T_N*T_L*T_W*T_H*unit;
  }
  if (Tank_Type->ItemIndex==0)
  {
  tank_vol->Text=FloatToStr(PI
          *(atof(Tsystem_design::tank_wd->Text.c_str())/2.0)
       *(atof(Tsystem_design::tank_wd->Text.c_str())/2.0)
       *(atof(Tsystem_design::tank_ht->Text.c_str()))*unit);
  T_V=PI
          *(atof(Tsystem_design::tank_wd->Text.c_str())/2.0)
       *(atof(Tsystem_design::tank_wd->Text.c_str())/2.0)
       *(atof(Tsystem_design::tank_ht->Text.c_str()))*unit;
  }

}
//---------------------------------------------------------------------------
```

err_F.cpp

```
//---------------------------------------------------------------------------
```

```
#include <vcl\vcl.h>
#pragma hdrstop

#include "crr_F.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TError_F *Error_F;
//---------------------------------------------------------------------------
__fastcall TError_F::TError_F(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TError_F::Button1Click(TObject *Sender)
{
        Close();
}
//---------------------------------------------------------------------------
```

filter_design.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "filter_design.h"
#define PI    3.14159265359
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
Tfilter_form *filter_form;
float p_dia,media_wt,s_g,total_surface_area;
char * media[10];
float calc_p_size(float);
//---------------------------------------------------------------------------
__fastcall Tfilter_form::Tfilter_form(TComponent* Owner)
        : TForm(Owner)
{

}
//---------------------------------------------------------------------------
void __fastcall Tfilter_form::Part_sizeClick(TObject *Sender)
{
        if (Part_size->ItemIndex==0)  p_dia=calc_p_size(30.0);
   if (Part_size->ItemIndex==1)  p_dia=calc_p_size(50.0);
   if (Part_size->ItemIndex==2)  p_dia=calc_p_size(70.0);
   if (Part_size->ItemIndex==3)  p_dia=calc_p_size(90.0);
    Show_p_size->Text=FloatToStr(p_dia);
   if (Part_size->ItemIndex==4)  Show_p_size->Text="Input value";


}
//---------------------------------------------------------------------------
```

```
float calc_p_size(float p_z)
{
  float size_mm;

  size_mm=(2.138E-6*p_z*p_z)-(8.026E-4*p_z)+(21.123*(1/(p_z+1)));
  return(size_mm);

}
void __fastcall Tfilter_form::Show_p_sizeChange(TObject *Sender)
{
   p_dia=atof(Tfilter_form::Show_p_size->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall Tfilter_form::calcClick(TObject *Sender)
{
  float part_vol,part_weight,num_per_g,area_per_g,media_wt,media_vol,v_ratio;


  //find volume of particle
  part_vol=(4.0/3.0)*PI*(pow((p_dia/2.0),3));
  //find the weight of the particle
  part_weight=(part_vol*s_g)/1000.0;
  //find number of particles per gram of media
  num_per_g=1.0/part_weight;
  //find the total area (cm) per gram
  area_per_g=(4.0*PI*(p_dia/2.0)*(p_dia/2.0)*num_per_g)/100.0; //divide by 100 for cm
  //find the Total Surface Area per weight of media
   media_wt=atof(Tfilter_form::weight_of_media->Text.c_str());
  total_surface_area=area_per_g*media_wt;
  //find the volume of the media with a given void ratio
  v_ratio=atof(Tfilter_form::V_R->Text.c_str());
  if (v_ratio != 0) media_vol=(media_wt/s_g)+((media_wt*v_ratio)/s_g);
          else Application->MessageBox("Need void Ratio", "Input Needed", MB_OK);

  TSA->Text=FloatToStr(total_surface_area);
  MV->Text=FloatToStr(media_vol);
}
//---------------------------------------------------------------------------
void __fastcall Tfilter_form::weight_of_mediaChange(TObject *Sender)
{
          media_wt=atof(Tfilter_form::weight_of_media->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall Tfilter_form::MediaClick(TObject *Sender)
{
   if (Media->ItemIndex==0) {media_type->Text="Sand";s_g=2.65;S_G->Text=2.65;}
   if (Media->ItemIndex==1) {media_type->Text="Plastic";S_G->Text="Input Value";}
   if (Media->ItemIndex==2) {media_type->Text="Enter Type";S_G->Text="Input Value";}


}
//---------------------------------------------------------------------------
void __fastcall Tfilter_form::V_RChange(TObject *Sender)
{
```

```
  V_R->Color=clWhite;
}
//---------------------------------------------------------------------------
```

graph_output.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include <math.h>
#include "Graph_Output.h"
#include "cfxocx.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
#define PI 3.141592
TVisual_Output *Visual_Output;
extern   struct AguaLog{
         int Water_Pump;
   int Air_Pump;
   int    MAN_PWM0;
   float FR_water;
   float FR_Set;
   float A_Press;
   float A_Set;
   float F_Press;
   float Filter_C_set;
   int  Sys_Water_Temp;
   int  Water_Temp_Set;
   int  Air_Temp;
   float DO;
   float DO_Set;
   float Total_Feed;
   int  FD_update_t;
   bool  L_Status;
   bool  H_Status;
   int     t_s;
   int     t_m;
   int     t_h;
   int     d_d;
   int  d_m;
   int  d_y;
   int  Update_t;
   int  Error;
   }AD;


//---------------------------------------------------------------------------
__fastcall TVisual_Output::TVisual_Output(TComponent* Owner)
        : TForm(Owner)
{


}
```

```
//---------------------------------------------------------------------------
void __fastcall TVisual_Output::Scan_PumpTimer(TObject *Sender)
{
        char sNum[40];
  Water_Pump_PWM->Position=AD.Water_Pump;
  WP_PWM->Caption=IntToStr(AD.Water_Pump);
  Air_Pump_PWM->Position=AD.Air_Pump;
  AP_PWM->Caption=IntToStr(AD.Air_Pump);

  if (nScrollLegend) {
     // delete the first legend
     W_Pump->Legend[0] = (char) 1;
     Air_P->Legend[0] = (char) 1;
     Diff_Press->Legend[0] = (char) 1;
     Temp_vs_Set->Legend[0] = (char) 1;
     if (!((nCounter+1) % nStep)) {
       wsprintf(sNum,"P %d",nCounter+1);
        W_Pump->Legend[nPoints - 1] = sNum;
        Air_P->Legend[nPoints - 1] = sNum;
        Diff_Press->Legend[nPoints - 1] = sNum;
        Temp_vs_Set->Legend[nPoints - 1] = sNum;
     } else
        W_Pump->Legend[nPoints - 1] = "";
         Air_P->Legend[nPoints - 1] = "";
         Diff_Press->Legend[nPoints - 1] = "";
         Temp_vs_Set->Legend[nPoints - 1] = "";
}

  W_Pump->OpenDataEx(COD_VALUES | COD_ADDPOINTS,2, 1);
  W_Pump->ThisSerie = 0;
  W_Pump->Value[0] = AD.FR_Set;
  W_Pump->ThisSerie = 1;
  W_Pump->Value[0] = AD.FR_water;
  W_Pump->CloseData((short) (COD_VALUES | COD_REALTIME | nScrollLegend));

  Air_P->OpenDataEx(COD_VALUES | COD_ADDPOINTS,4, 1);
  Air_P->ThisSerie = 0;
  Air_P->Value[0] = AD.A_Set;
  Air_P->ThisSerie = 1;
  Air_P->Value[0] = AD.A_Press;
  Air_P->ThisSerie = 2;
  Air_P->Value[0] = AD.DO_Set;
  Air_P->ThisSerie = 3;
  Air_P->Value[0] = AD.DO;

  Air_P->CloseData((short) (COD_VALUES | COD_REALTIME | nScrollLegend));

  Diff_Press->OpenDataEx(COD_VALUES | COD_ADDPOINTS,2, 1);
  Diff_Press->ThisSerie = 0;
  Diff_Press->Value[0] = AD.Filter_C_set;
  Diff_Press->ThisSerie = 1;
  Diff_Press->Value[0] = AD.F_Press;
  Diff_Press->CloseData((short) (COD_VALUES | COD_REALTIME | nScrollLegend));
```

```
  Temp_vs_Set->OpenDataEx(COD_VALUES | COD_ADDPOINTS,3, 1);
  Temp_vs_Set->ThisSerie = 0;
  Temp_vs_Set->Value[0] = AD.Water_Temp_Set;
  Temp_vs_Set->ThisSerie = 1;
  Temp_vs_Set->Value[0] = AD.Sys_Water_Temp;
  Temp_vs_Set->ThisSerie = 2;
  Temp_vs_Set->Value[0] = AD.Air_Temp;
  Temp_vs_Set->CloseData((short) (COD_VALUES | COD_REALTIME | nScrollLegend));

  nCounter = nCounter + 1;
}
//---------------------------------------------------------------------------
void __fastcall TVisual_Output::FormCreate(TObject *Sender)
{
  int i;
  char sNum[80];

  W_Pump->Type_ |= CT_EVENSPACING | CT_SHOWZERO;
  W_Pump->RealTimeStyle = CRT_NOWAITARROW;
  Air_P->Type_ |= CT_EVENSPACING | CT_SHOWZERO;
  Air_P->RealTimeStyle = CRT_NOWAITARROW;
  Diff_Press->Type_ |= CT_EVENSPACING | CT_SHOWZERO;
  Diff_Press->RealTimeStyle = CRT_NOWAITARROW;
  Temp_vs_Set->Type_ |= CT_EVENSPACING | CT_SHOWZERO;
  Temp_vs_Set->RealTimeStyle = CRT_NOWAITARROW;

  // Improve speed
  Air_P->StyleEx |= CSE_FASTLEGENDS;
  Diff_Press->StyleEx |= CSE_FASTLEGENDS;
  W_Pump->StyleEx |= CSE_FASTLEGENDS;
  Temp_vs_Set->StyleEx |= CSE_FASTLEGENDS;

  // Buffer Size
  nPoints = 30;
  nStep = 5;
  Air_P->MaxValues = nPoints;
  Diff_Press->MaxValues = nPoints;
  W_Pump->MaxValues = nPoints;
  Temp_vs_Set->MaxValues = nPoints;

  // Initial chart
  W_Pump->OpenDataEx(COD_VALUES,2,nPoints);
  W_Pump->CloseData(COD_VALUES);
  Air_P->OpenDataEx(COD_VALUES,4,nPoints);
  Air_P->CloseData(COD_VALUES);
  Diff_Press->OpenDataEx(COD_VALUES,2,nPoints);
  Diff_Press->CloseData(COD_VALUES);
  Temp_vs_Set->OpenDataEx(COD_VALUES,3,nPoints);
  Temp_vs_Set->CloseData(COD_VALUES);

  // Legend
  for (i = 0; i < nPoints; i++)
    if (!((i+1) % nStep)) {
      wsprintf(sNum,"P %d",i+1);
```

```
        W_Pump->Legend[i] = sNum;
        Air_P->Legend[i] = sNum;
        Diff_Press->Legend[i] = sNum;
        Temp_vs_Set->Legend[i] = sNum;
    }

    W_Pump->LegStyle |= CL_NOTCLIPPED;
    Air_P->LegStyle |= CL_NOTCLIPPED;
    Diff_Press->LegStyle |= CL_NOTCLIPPED;
    Temp_vs_Set->LegStyle |= CL_NOTCLIPPED;

    // Grid preferences
    W_Pump->VertGridGap = (short) nStep;
    W_Pump->Grid = CHART_VERTGRID;
    W_Pump->ItemColor[CI_VERTGRID] = (TColor) RGB(0, 128, 0);
    Air_P->VertGridGap = (short) nStep;
    Air_P->Grid = CHART_VERTGRID;
    Air_P->ItemColor[CI_VERTGRID] = (TColor) RGB(0, 128, 0);
    Diff_Press->VertGridGap = (short) nStep;
    Diff_Press->Grid = CHART_VERTGRID;
    Diff_Press->ItemColor[CI_VERTGRID] = (TColor) RGB(0, 128, 0);
    Temp_vs_Set->VertGridGap = (short) nStep;
    Temp_vs_Set->Grid = CHART_VERTGRID;
    Temp_vs_Set->ItemColor[CI_VERTGRID] = (TColor) RGB(0, 128, 0);


    // Loop Preferences
    W_Pump->ItemColor[CI_LOOPPOS] = (TColor) RGB(255, 0, 0);
    Air_P->ItemColor[CI_LOOPPOS] = (TColor) RGB(255, 0, 0);
    Diff_Press->ItemColor[CI_LOOPPOS] = (TColor) RGB(255, 0, 0);
    Temp_vs_Set->ItemColor[CI_LOOPPOS] = (TColor) RGB(255, 0, 0);

    // Y Axis preferences
    W_Pump->Adm[CSA_GAP] = .1;
    W_Pump->Adm[CSA_MIN] = 0;
    W_Pump->Adm[CSA_MAX] = 2.0;
    Air_P->Adm[CSA_GAP] = .1;
    Air_P->Adm[CSA_MIN] = 0;
    Air_P->Adm[CSA_MAX] = 10;
    Diff_Press->Adm[CSA_GAP] = .1;
    Diff_Press->Adm[CSA_MIN] = 0;
    Diff_Press->Adm[CSA_MAX] = 10;
    Temp_vs_Set->Adm[CSA_GAP] = 1;
    Temp_vs_Set->Adm[CSA_MIN] = 10;
    Temp_vs_Set->Adm[CSA_MAX] = 30;

    nCounter = nPoints;
    nScrollLegend = 0;

    W_Pump->ItemWidth[CI_2DLINE] = 1;
    Air_P->ItemWidth[CI_2DLINE] = 1;
    Diff_Press->ItemWidth[CI_2DLINE] = 1;
    Temp_vs_Set->ItemWidth[CI_2DLINE] = 1;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TVisual_Output::CheckBox2Click(TObject *Sender)
{
/* if (CheckBox2->Checked) {
      nScrollLegend = COD_SCROLLLEGEND;
      W_Pump->TypeEx |= CTE_NOLEGINVALIDATE;
      Air_P->TypeEx |= CTE_NOLEGINVALIDATE;
      Diff_Press->TypeEx |= CTE_NOLEGINVALIDATE;
      DO_vs_Set->TypeEx |= CTE_NOLEGINVALIDATE;
   } else {
      nScrollLegend = 0;
      W_Pump->TypeEx &= ~CTE_NOLEGINVALIDATE;
      Air_P->TypeEx &= ~CTE_NOLEGINVALIDATE;
      Diff_Press->TypeEx &= ~CTE_NOLEGINVALIDATE;
      DO_vs_Set->TypeEx &= ~CTE_NOLEGINVALIDATE;
   }  */

}

//---------------------------------------------------------------------------
void __fastcall TVisual_Output::CheckBox1Click(TObject *Sender)
{
/*  // Scroll all the way to the end
   W_Pump->Scroll(SB_BOTTOM,0);
   W_Pump->Scroll(SB_ENDSCROLL,0);
   Air_P->Scroll(SB_BOTTOM,0);
   Air_P->Scroll(SB_ENDSCROLL,0);
   Diff_Press->Scroll(SB_BOTTOM,0);
   Diff_Press->Scroll(SB_ENDSCROLL,0);
   DO_vs_Set->Scroll(SB_BOTTOM,0);
   DO_vs_Set->Scroll(SB_ENDSCROLL,0);

   if (CheckBox1->Checked)

      Scan_Pump->Interval = 2000;

   else
      Scan_Pump->Interval = 0;
  */
}
//---------------------------------------------------------------------------
void __fastcall TVisual_Output::CheckBox3Click(TObject *Sender)
{
/*   if (CheckBox3->Checked)
      W_Pump->RealTimeStyle |= CRT_LOOPPOS;
   else
      W_Pump->RealTimeStyle &= ~CRT_LOOPPOS;
   // Refresh the data
   W_Pump->Refresh();   */
}
//---------------------------------------------------------------------------
void __fastcall TVisual_Output::CheckBox6Click(TObject *Sender)
{
 /*        if (CheckBox6->Checked)
```

```
    Air_P->RealTimeStyle |= CRT_LOOPPOS;
  else
    Air_P->RealTimeStyle &= ~CRT_LOOPPOS;
  // Refresh the data
  Air_P->Refresh();
  */
}
//---------------------------------------------------------------------------
void __fastcall TVisual_Output::FormActivate(TObject *Sender)
{
   // Scroll all the way to the end
   W_Pump->Scroll(SB_BOTTOM,0);
   W_Pump->Scroll(SB_ENDSCROLL,0);
   Air_P->Scroll(SB_BOTTOM,0);
   Air_P->Scroll(SB_ENDSCROLL,0);
   Diff_Press->Scroll(SB_BOTTOM,0);
   Diff_Press->Scroll(SB_ENDSCROLL,0);
   Temp_vs_Set->Scroll(SB_BOTTOM,0);
   Temp_vs_Set->Scroll(SB_ENDSCROLL,0);
   Scan_Pump->Interval = 2000;
}
//---------------------------------------------------------------------------
```

photo.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "photo.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <dos.h>
#include <vcl/dstring.h>



#define TRUE  1
#define FALSE 0
#define PI    3.14159265359          //1 radian = 57.29577951 degrees
#define RAD       57.29577951                    //1 degree = .01745329252 radians
#define deg   .01745329252

void  calc_ET(int day);
void  calc_AST(float LST);
void  calc_hr_a();
void  calc_dec_a(int day);
void  calc_sol_alt();
```

```
void   calc_sol_az();
void   calc_sr_ss();
void   solar();
void   calc_year_day(void);

int i,hr,day=0;
bool light;
float Lat,Long,start_time;
float hour_a,dec_a,sol_alt; //hour angle,declanation angle,solar altitude
float sol_az;              //solar azimuth
float AST;                 //apparent solar time,local standard time
float LSM,LST=0,LST_min=0;          //Local standard time meridian, degrees
float ET;                  //equation of time, minuites
float hs,t_sunrise,t_sunset,length_of_day,tsmin,tss;    //sunset hour angle, sunrise time,sunset time
float tilt=0;              // angle of surface sun impinges on it is horizontal


struct time t;
struct date dt;


enum
{
        JAN=1,
   FEB,
   MAR,
   APR,
   MAY,
   JUN,
   JUL,
   AUG,
   SEP,
   OCT,
   NOV,
   DEC
};

TphotoF *photoF;
//---------------------------------------------------------------------------
__fastcall TphotoF::TphotoF(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------


//----------------------------functions to calculate suns position---------------

 void calc_ET(int day)   //eqation of time in minuites
{
```

```
        ET=9.87*sin( ( 4.0*PI* ( (day-81.0) /364.0) ) )
                        -7.53*cos( (2.0*PI* ((day-81.0)/364.0) ))
                        -1.5*sin((2.0*PI*((day-81.0)/364.0)));
}

void calc_AST(float LST)  //calcs apparent solar time
{
 float temp;

 calc_ET(day);
 LSM=floor(Long/15)*15.0;
 temp=((ET+(4.0*(LSM-Long)))/60.0);
 AST=LST+temp;
}

void calc_hr_a()    //calculates hour angle
{
        hour_a=(AST-12.0)*15.0;
}

void calc_dec_a(int day)   //calculates declination angle
{
        dec_a=23.45*sin(360.0*((284.0+day)/365.0)/RAD);
}

void calc_sol_alt()       //calculates solar altitude
{
        sol_alt=asin(cos(Lat/RAD)*cos(dec_a/RAD)\
        *cos(hour_a/RAD)+(sin(Lat/RAD)*sin(dec_a/RAD)))*(180.0/PI);
}

void calc_sol_az()        //calculates solar azimuth
{
        if( (sin(sol_alt/RAD)*sin(Lat/RAD)-sin(dec_a/RAD)/(cos(sol_alt/RAD)*cos(Lat/RAD))) >= -1
&&\
    (sin(sol_alt/RAD)*sin(Lat/RAD)-sin(dec_a/RAD)/(cos(sol_alt/RAD)*cos(Lat/RAD))) <= 1)
        sol_az=acos((sin(sol_alt/RAD)*sin(Lat/RAD)-sin(dec_a/RAD))\
        /(cos(sol_alt/RAD)*cos(Lat/RAD)))*(hour_a/fabs(hour_a))*(180.0/PI);
   else
   sol_az=0;
}


void calc_sr_ss()        //calc sunrise and sunset
{
   int found=0;
   float min,tday_sr;

 for(min=0;min<=1440;min++)
 {
   calc_AST(min/60);
   calc_hr_a();
        calc_dec_a(day);
        calc_sol_alt();
```

```
        if(sol_alt>=0 && found==0)
    {
        t_sunrise=(floor(min/60)*100)+(int)min%60;
      tday_sr=min;
      found=1;
    }
    if(sol_alt<=0 && found==1)
    {
        t_sunset=(floor(min/60)*100)+(int)min%60;
      length_of_day=(floor(((min-tday_sr)/60))*100)+(int)(min-tday_sr)%60;
      return;
    }
  }
}

void solar()
{
 gettime(&t);    //get the present time (put into structure t)
 LST_min=t.ti_min+(t.ti_hour*60);  //calc present time in minites
 LST=LST_min/60;      //change to fractional hours

 calc_year_day();  //calc the day of the year

 calc_AST(LST);
 calc_hr_a();
 calc_dec_a(day);
 calc_sol_alt();
 calc_sol_az();
 }
//---------------------------------------------------------------------------

void __fastcall TphotoF::Timer1Timer(TObject *Sender)
{


 Lat=atof(TphotoF::latitude->Text.c_str());     //latitude
 Long=atof(TphotoF::longitude->Text.c_str());   //longitude
 date_month->Caption= FloatToStr(dt.da_mon);
 date_day->Caption= FloatToStr(dt.da_day);
 date_year->Caption= FloatToStr(dt.da_year);
 day_y->Caption=IntToStr(day);
 time_hr->Caption= FloatToStr(t.ti_hour);

 solar();  //calculates the suns position

 if(sol_alt>0)
 { sun_pos->Caption="Day";
  sun_pos->Color=clYellow;
  light=true;
 }
 else
 {sun_pos->Caption="Night";
  sun_pos->Color=clBlack;
  light=false;
```

```
 }

String wd=IntToStr(t.ti_min);

// int ln=strlen((char) wd->Text.c_str());
// if(()==1)time_min->Caption=StrCat(0,FloatToStr(t.ti_min));
// else
 time_min->Caption=FloatToStr(t.ti_min);

 calc_sr_ss();
 _sunrise->Text=FloatToStr(t_sunrise);     //time of sunrise t_sunrise
 _sunset->Text=FloatToStr(t_sunset);       //time of sunset  t_sunset
 _day_length->Text=FloatToStr(length_of_day);    //total day length



}
//---------------------------------------------------------------------------
void calc_year_day(void)
{
        int m_day;
getdate(&dt);   //get the present date (put into structure dt)

switch(dt.da_mon)
{
        case JAN:m_day=0;break;
  case FEB:m_day=31;break;
  case MAR:m_day=59;break;
  case APR:m_day=89;break;
  case MAY:m_day=120;break;
  case JUN:m_day=150;break;
  case JUL:m_day=181;break;
  case AUG:m_day=212;break;
  case SEP:m_day=242;break;
  case OCT:m_day=273;break;
  case NOV:m_day=303;break;
  case DEC:m_day=334;break;

}

day=m_day+ dt.da_day;  //calc the day of the year

}


//---------------------------------------------------------------------------
```

pump_error.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "pump_error.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
bool COND_NOTED=false;
TPUMP_ERR *PUMP_ERR;
//---------------------------------------------------------------------------
__fastcall TPUMP_ERR::TPUMP_ERR(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TPUMP_ERR::BitBtn1Click(TObject *Sender)
{
  COND_NOTED=true;
  pump_err_timer->Enabled=true;
        Close();
}
//---------------------------------------------------------------------------
void __fastcall TPUMP_ERR::pump_err_timerTimer(TObject *Sender)
{
  COND_NOTED=false;
  pump_err_timer->Enabled=false;
}
//---------------------------------------------------------------------------
```

species.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "species.h"
#include "_carp.h"
#include "_Trout.h"
#include "_Salmon.h"
#include "_Stripped_Bass.h"
#include "_Tilapia.h"
#include "_Catfish.h"
#include "_Sturgeon.h"
#include "_Tropical.h"
#include "_oxygen.h"
#include "growth.h"
```

```
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
Tselect_species *select_species;
int Tank_Number;

extern struct growth_data
{
   int  tank_num:       //tank number
   int  num_fish:       //number of fish in tank
        float sp_gr_fac:      //growth factor
   float sp_b:          //b factor
   float sp_fin_wt:     //final expected wt
   float begin_wt:      //begining wt
   float present_wt:    //present weight
   float market_weight:  //market weight of fish
   float feed_rate:     //% body weight to feed at
   int  Start_Day:     //day culture was spawned
   int  Grow_Out_days:  //number of days until harvest size
   int  Days_Of_Culture:  //number of days the culture
   char* sp_name;

}T[4];
        char* sp_Carp="Carp";
        char* sp_Tilapia="Tilapia";
        char* sp_Catfish="Catfish";
        char* sp_Salmon="Salmon";
        char* sp_Stripped_Bass="Stripped Bass";
        char* sp_Sturgeon="Sturgeon";
        char* sp_Tropical="Tropical";
        char* sp_Trout="Trout";

//---------------------------------------------------------------------------
__fastcall Tselect_species::Tselect_species(TComponent* Owner)
        : TForm(Owner)
{

}
//---------------------------------------------------------------------------
void __fastcall Tselect_species::closeClick(TObject *Sender)
{
        Close();
}
//---------------------------------------------------------------------------
void __fastcall Tselect_species::species_parametersClick(TObject *Sender)
{
        if (species_parameters->ItemIndex==0) {T[Tank_Number].sp_name=sp_Carp;Carp->Show();}
   if (species_parameters->ItemIndex==1) {T[Tank_Number].sp_name=sp_Catfish;Catfish->Show();}
   if (species_parameters->ItemIndex==2) {T[Tank_Number].sp_name=sp_Salmon;Salmon->Show();}
   if (species_parameters->ItemIndex==3) {T[Tank_Number].sp_name=sp_Stripped_Bass;Stripped_Bass-
>Show();}
   if (species_parameters->ItemIndex==4) {T[Tank_Number].sp_name=sp_Sturgeon;Sturgeon->Show();}
   if (species_parameters->ItemIndex==5) {T[Tank_Number].sp_name=sp_Tilapia;Tilapia->Show();}
   if (species_parameters->ItemIndex==6) {T[Tank_Number].sp_name=sp_Tropical;Tropical->Show();}
   if (species_parameters->ItemIndex==7) {T[Tank_Number].sp_name=sp_Trout;Trout->Show();}
```

```
}
//---------------------------------------------------------------------------
void __fastcall Tselect_species::T_N_CClick(TObject *Sender,
      TUDBtnType Button)
{
      T_N->Text=T_N_C->Position;
  Tank_Number=atoi(Tselect_species::T_N->Text.c_str())-1;
  T[Tank_Number].tank_num=Tank_Number;
  species_parameters->ItemIndex=-1;
}
//---------------------------------------------------------------------------
void __fastcall Tselect_species::FormClose(TObject *Sender,
      TCloseAction &Action)
{
      species_parameters->ItemIndex=-1;
}
//---------------------------------------------------------------------------
void __fastcall Tselect_species::FormActivate(TObject *Sender)
{
      Tank_Number=(T_N_C->Position)-1;

}
//---------------------------------------------------------------------------
```

waste.cpp

```
//This section decides when the water is to be pumped from the system and
//how much is to be pumped. The requirements for this to work are the
//flow rate of the waste pump,
//the % of water to be wasted, (need to know how much water in system)
//the time interval between pumping (min) IBE
//W_Water=W_M_S.DER*Tot_Sys_Water
//N_D=1440/IBE              Number of Dumps


//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "Waste.h"
#include <time.h>
#include <dos.h>
#include <math.h>
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TW_M *W_M;

bool D_WASTE;
extern struct time t;
extern struct date dt;
```

```
extern float T_V;
extern int D_Time;
int T_B_D;          //time between dumps (min)
float W_P_D;        //time per dump (sec)
int T_P_D;
struct Waste_M
{
 float PFR;                 //waste pump flow rate in gpm
 float DER;                 //daily exchange rate % of total volume of system
 int   IBE;                 //time interval between wasting of water from system
 int   TPO;                 //time pump will be turned on sec
 float NO3L;        //last entered nitrate level
 float NO2L;                //last entered nitrite level
 int       t_s;             //time seconds
 int   t_m;         //time min
 int       t_h;             //time hours
 int       d_d;             //date day
 int       d_m;             //date month
 int       d_y;             //date year
}W_M_L,W_M_S;

void Dump_Time(void);
//---------------------------------------------------------------------------
__fastcall TW_M::TW_M(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TW_M::FormActivate(TObject *Sender)
{
//
        Dump_Time();    //calc next dump time

  PFR->Text=FloatToStr(W_M_S.PFR);
  DER->Text=FloatToStr(W_M_S.DER);
  IBE->Text=IntToStr(W_M_S.IBE);
  T_P_O->Text=FloatToStr(W_M_S.TPO);
  NO3L->Text=FloatToStr(W_M_S.NO3L);
  NO2L->Text=FloatToStr(W_M_S.NO2L);

  Dump_Time();
}
//---------------------------------------------------------------------------
void __fastcall TW_M::PFRChange(TObject *Sender)
{
 W_M_S.PFR=atof(TW_M::PFR->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TW_M::DERChange(TObject *Sender)
{
 W_M_S.DER=atof(TW_M::DER->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TW_M::IBEChange(TObject *Sender)
```

```
{
  W_M_S.IBE=atof(TW_M::IBE->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TW_M ::NO3LChange(TObject *Sender)
{
  W_M_S.NO3L=atof(TW_M::NO3L->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TW_M::NO2LChange(TObject *Sender)
{
  W_M_S.NO2L=atof(TW_M::NO2L->Text.c_str());
}
//---------------------------------------------------------------------------
void Dump_Time(void)
{
  int N_D;
  float W_Water,Tot_Sys_Water;

  Tot_Sys_Water=T_V;

  W_Water=W_M_S.DER*Tot_Sys_Water*.01;
       // amount of water to be wasted in a day (gal)
  N_D=1440/W_M_S.IBE;                    //number of times in a day the water is wasted
  W_P_D=W_Water/(float)N_D;      //gals to be pumped per dump

  while(W_P_D<.17)   // minimun amount to be pumped out is 1/6 gal
  {
         W_M_S.IBE++;
   N_D=1440/W_M_S.IBE;
   W_P_D=W_Water/(float)N_D;   //

  }

  T_B_D=1440/N_D;                    //time between dumps
  W_M_S.TPO=T_P_D=(int)ceil(W_P_D/(W_M_S.PFR/60));          //time (sec) for each dumping

}

void __fastcall TW_M::Timer1Timer(TObject *Sender)
{  //checks every minute to see if dump of waste should occur
   int time_now,hr,min;

         gettime(&t);
   time_now=t.ti_min + (t.ti_hour*60);
   T_B_D=W_M_S.IBE;
   if(time_now%T_B_D==0)
         {
         D_WASTE=true;
        Timer1->Enabled=false;  //disables timer1
        Timer2->Enabled=true;   //starts timer 2

      }
```

```
  while(time_now%T_B_D!=0){time_now++;}//calc the next time to dump
  hr=time_now/60;                          //hrs
  min=time_now-((time_now/60)*60);  //min
  Label15->Caption=IntToStr(hr);
  Label17->Caption=IntToStr(min);
}
//---------------------------------------------------------------------------
void __fastcall TW_M::Timer2Timer(TObject *Sender)
{
    Timer1->Enabled=true;  //restart timer1
}
//---------------------------------------------------------------------------


void __fastcall TW_M::StartClick(TObject *Sender)
{
        Dump_Time(); //calculats dump times

  IBE->Text=FloatToStr(W_M_S.IBE);
  T_P_O->Text=IntToStr(W_M_S.TPO);
//  Timer1->Enabled=true;
}
//---------------------------------------------------------------------------

void __fastcall TW_M::File_SaveClick(TObject *Sender)
{


  FILE * f;          //writes to the waste log file
  f= fopen ("WASTE_LOG.TXT", "at");
  if (f==NULL)return;

  gettime(&t);
  getdate(&dt);
  W_M_L.t_s = t.ti_sec;
  W_M_L.t_m = t.ti_min;
  W_M_L.t_h = t.ti_hour;
  W_M_L.d_d = dt.da_day;
  W_M_L.d_m = dt.da_mon;
  W_M_L.d_y = dt.da_year;

  fprintf(f,"%4.2f\t%4.2f\t%d\t%d\t%4.2f\t%4.2f\t%d\t%d\t%d\t%d\t%d\t%d",

        W_M_L.PFR,W_M_L.DER,W_M_L.IBE,W_M_L.TPO,W_M_L.NO3L,W_M_L.NO2L,
       W_M_L.t_s,W_M_L.t_m,W_M_L.t_h,W_M_L.d_d,W_M_L.d_m,W_M_L.d_y);
  fprintf(f, "\n");
  fclose(f);
//--------writes to the waste status file
  FILE * g;
  g= fopen ("WASTE_STATUS.TXT", "w");
  if (g==NULL)return;
  fprintf(g,"%4.2f %4.2f %d %d %4.2f %4.2f %d %d %d %d %d %d",
                W_M_S.PFR,W_M_S.DER,W_M_S.IBE,W_M_S.TPO,W_M_S.NO3L,W_M_S.NO2L,
       W_M_S.t_s,W_M_S.t_m,W_M_S.t_h,W_M_S.d_d,W_M_S.d_m,W_M_S.d_y);
```

```
  fclose(g);
}
//---------------------------------------------------------------------------
```

Comport.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "ComPort.h"
//---------------------------------------------------------------------------
static inline TComPort *ValidCtrCheck()
{
        return new TComPort(NULL);
}
//---------------------------------------------------------------------------
namespace Comport
{
        void __fastcall Register()
        {
                TComponentClass classes[1] = {__classid(TComPort)};
                RegisterComponents("Samples", classes, 0);
        }
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
const char * TComPort::PortName [5] = { "NUL", "COM1", "COM2", "COM3", "COM4" };
//---------------------------------------------------------------------------
// The constructor for TComPort objects sets the default values for properties, creates
// event objects to handle overlapped IO and initializes the overlapped IO structures:
__fastcall TComPort::TComPort(TComponent* Owner)
        : TComponent(Owner)
{
        BaudRate = 9600;
        ByteSize = _8_Bits;
        Parity = NoParity;
        StopBits = _1_Stop;
        BufferSize = DEFAULT_BUFFER_SIZE;
        EndOfLineCharacter = '\n';
        hComPort = INVALID_HANDLE_VALUE;
        ReadComPortLoop = NULL;
        WriteComPortLoop = NULL;
        InputBuffer = NULL;
        OutputBuffer = NULL;
        hWriteComPortDone = CreateEvent (
                NULL, // event object handle cannot be inherited
```

```
            TRUE, // manual reset
            FALSE, // initial state is nonsignaled
            NULL // create the event object with no name
        );
        if (hWriteComPortDone == NULL) { // if unable to open event object handle
            hReadComPortDone = NULL;
            hCharactersToWrite = NULL;
            return; // exit quietly
        }
        hReadComPortDone = CreateEvent (
            NULL, // event object handle cannot be inherited
            TRUE, // manual reset
            FALSE, // initial state is nonsignaled
            NULL // create the event object with no name
        );
        if (hReadComPortDone == NULL) { // if unable to open event object handle
            CloseHandle (hWriteComPortDone);
            hWriteComPortDone = NULL;
            hCharactersToWrite = NULL;
            return; // exit quietly
        }
        hCharactersToWrite = CreateEvent (
            NULL, // event object handle cannot be inherited
            TRUE, // manual reset
            FALSE, // initial state is nonsignaled
            NULL // create the event object with no name
        );
        if (hCharactersToWrite == NULL) { // if unable to open event object handle
            CloseHandle (hReadComPortDone);
            CloseHandle (hWriteComPortDone);
            hReadComPortDone = NULL;
            hWriteComPortDone = NULL;
            return; // exit quietly
        }
        ComPortWriteOL.Offset = 0; // the offset must be zero
        ComPortWriteOL.OffsetHigh = 0; // the offset must be zero
        ComPortWriteOL.hEvent = hWriteComPortDone; // Event handle
        ComPortReadOL.Offset = 0; // the offset must be zero
        ComPortReadOL.OffsetHigh = 0; // the offset must be zero
        ComPortReadOL.hEvent = hReadComPortDone; // Event handle
}
//---------------------------------------------------------------------------
// The destructor for TComPort objects closes the comm port if it is open and destroys the event
// objects that handle overlapped IO:
__fastcall TComPort::~TComPort() {
        // Make sure the port is closed before destroying the object:
        Close ();
        // Close the wait for cgaracters to write event object handle:
        if (hCharactersToWrite)
            CloseHandle (hCharactersToWrite);
        // Close the write event object handle:
        if (hWriteComPortDone)
            CloseHandle (hWriteComPortDone);
        // Close the read event object handle:
```

```
        if (hReadComPortDone)
                CloseHandle (hReadComPortDone);
}
//---------------------------------------------------------------------------
void __fastcall TComPort::SetBaudRate (DWORD value) {
        if (value >= 256000) FBaudRate = 256000;
        else if (value >= 128000) FBaudRate = 128000;
        else if (value >= 115200) FBaudRate = 115200;
        else if (value >= 57600) FBaudRate = 57600;
        else if (value >= 56000) FBaudRate = 56000;
        else if (value >= 38400) FBaudRate = 38400;
        else if (value >= 19200) FBaudRate = 19200;
        else if (value >= 14400) FBaudRate = 14400;
        else if (value >= 9600) FBaudRate = 9600;
        else if (value >= 4800) FBaudRate = 4800;
        else if (value >= 2400) FBaudRate = 2400;
        else if (value >= 1200) FBaudRate = 1200;
        else if (value >= 600) FBaudRate = 600;
        else if (value >= 300) FBaudRate = 300;
        else FBaudRate = 110;
}
//---------------------------------------------------------------------------
void __fastcall TComPort::SetByteSize (TByteSize value) {
        FByteSize = value;
        if ((value == _5_Bits) && (StopBits == _2_Stop))
                StopBits = _1_5_Stop;
        if ((value != _5_Bits) && (StopBits == _1_5_Stop))
                StopBits = _2_Stop;
}
//---------------------------------------------------------------------------
void __fastcall TComPort::SetStopBits (TStopBits value) {
        FStopBits = value;
        if ((ByteSize == _5_Bits) && (value == _2_Stop))
                ByteSize = _6_Bits;
        if ((ByteSize != _5_Bits) && (value == _1_5_Stop))
                ByteSize = _5_Bits;
}
//---------------------------------------------------------------------------
void __fastcall TComPort::Open (void) {
        DCB ComPortDCB;// device control block for the comm port
        if (! (hReadComPortDone && hWriteComPortDone)) { // if the event handlers were not properly
initialized
                Application->MessageBox ("Event handlers not initialized", "Communications error",
MB_OK); // display the error to the user
                return; // return without crashing
        }
        if (hComPort != INVALID_HANDLE_VALUE) // if there is a handle to the com port, the port is
open
                Close (); // in this case, close the com port first
        hComPort = CreateFile (
                PortName [Port], // file name for the comm port
                GENERIC_READ | GENERIC_WRITE, // open for read and write
                0, // do not allow sharing of the comm port
                NULL, // comm port file handle cannot be inherited
```

```
                OPEN_EXISTING, // open the existing comm port; fail if it does not exist
                FILE_ATTRIBUTE_NORMAL | // file attributes are normal
                FILE_FLAG_OVERLAPPED, // IO operations may return ERROR_IO_PENDING if
they take a long time:
                        // in this case, when the operation is finished it triggers an event; as a
consequence,
                        // the application must perfor overlapped reading and writing
                NULL // template file must be NULL
        );
        if (hComPort == INVALID_HANDLE_VALUE) { // if the comm port was not successfully open
                System::AnsiString Message = "Unable to open comm port: Error # ";
                Message += AnsiString ((int) GetLastError());
                Application->MessageBox (Message.c_str(), "Communications error", MB_OK); //
display the error to the user
                return; // return without crashing
        }
        if (! GetCommState (hComPort, &ComPortDCB)) { // read the current settings on the comm port
                AnsiString Message = "Unable to get comm port state: Error # ";
                Message += AnsiString ((int) GetLastError());
                CloseHandle (hComPort);
                hComPort = INVALID_HANDLE_VALUE;
                Application->MessageBox (Message.c_str(), "Communications error", MB_OK); //
display the error to the user
                return; // return without crashing
        }
        ComPortDCB.BaudRate = BaudRate; // 9600 baud
        ComPortDCB.ByteSize = (BYTE) (ByteSize + 5); // ByteSize starts at _5_bits = 0, so 5 must be
added
        ComPortDCB.Parity = Parity; // no parity (8N1)
        ComPortDCB.StopBits = StopBits; // one stop bit (8N1)
        if (! SetCommState (hComPort, &ComPortDCB)) { // set the comm port to the settings above
                AnsiString Message = "Unable to set comm port state: Error # ";
                Message += AnsiString ((int) GetLastError());
                CloseHandle (hComPort);
                hComPort = INVALID_HANDLE_VALUE;
                Application->MessageBox (Message.c_str(), "Communications error", MB_OK); //
display the error to the user
                return; // return without crashing
        }
        if (! SetupComm (hComPort, BufferSize, BufferSize)) { // set the sizes of the communication
buffers
                AnsiString Message = "Unable to set up comm port buffers: Error # ";
                Message += AnsiString ((int) GetLastError());
                CloseHandle (hComPort);
                hComPort = INVALID_HANDLE_VALUE;
                Application->MessageBox (Message.c_str(), "Communications error", MB_OK); //
display the error to the user
                return; // return without crashing
        }
        OpenBufferSize = BufferSize;
        InputBuffer = new char [OpenBufferSize];
        BytesRead = 0;
        OutputBuffer = new char [OpenBufferSize];
        NextByteOut = LastByteOut = 0;
```

```
        ReadComPortLoop = new TReadComPortLoop (this); // start the thread to read characters
        WriteComPortLoop = new TWriteComPortLoop (this); // start the thread to write characters
}
//---------------------------------------------------------------------------
void __fastcall TComPort::Close (void) {
        if (WriteComPortLoop)
                WriteComPortLoop->Terminate();
        WriteComPortLoop = NULL;
        if (ReadComPortLoop)
                ReadComPortLoop->Terminate();
        ReadComPortLoop = NULL;
        if (OutputBuffer)
                delete [] OutputBuffer;
        if (InputBuffer)
                delete [] InputBuffer;
        if (hComPort != INVALID_HANDLE_VALUE)
                CloseHandle (hComPort);
        hComPort = INVALID_HANDLE_VALUE;
}
//---------------------------------------------------------------------------
void __fastcall TComPort::Write (System::AnsiString Line) {
        // If the port is not open, do not write the line:
        if (hComPort == INVALID_HANDLE_VALUE)
                return;
        // Suspend the thread that writes to the comm port while the output buffer is being updated:
        WriteComPortLoop->Suspend();
        for (int i = 1; i <= Line.Length(); i ++) {
                OutputBuffer [LastByteOut] = Line [i];
                LastByteOut ++;
                if (LastByteOut == OpenBufferSize)
                        LastByteOut = 0;
                if (LastByteOut == NextByteOut) {
                        if (LastByteOut == 0)
                                LastByteOut = OpenBufferSize;
                        LastByteOut --;
                }
        }
        SetEvent (hCharactersToWrite);
        // Resume the thread that writes to the comm port after the output buffer has been updated:
        WriteComPortLoop->Resume();
}

//---------------------------------------------------------------------------
void __fastcall TComPort::ReadByte (void) {
        if (OnReadByte)
                OnReadByte (this, InputBuffer [BytesRead]);
}
//---------------------------------------------------------------------------
void __fastcall TComPort::ReadLine (void) {
        if (OnReadLine)
                OnReadLine (this, InputBuffer);
}
//---------------------------------------------------------------------------
void __fastcall TComPort::WriteByte (void) {
```

```
        if (OnWriteByte)
                OnWriteByte (this, OutputBuffer [NextByteOut]);
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
__fastcall TReadComPortLoop::TReadComPortLoop (TComPort * Port)
        : TThread(true)
{
        ComPort = Port;
        FreeOnTerminate = true; // delete the object automatically when the thread terminates
        Resume ();
}
//---------------------------------------------------------------------------
void __fastcall TReadComPortLoop::Execute()
{
        char LastByteRead;
        DWORD BytesRead;
        for (;;) {
                // If the thread should be terminated, abandon monitoring comm port input:
                if (Terminated)
                        return;
                // If the port is not open, terminate the thread:
                if (ComPort->hComPort == INVALID_HANDLE_VALUE)
                        return;
                // Try to read one character from the comm port:
                if (! ReadFile (ComPort->hComPort, &LastByteRead, 1, &BytesRead, &(ComPort-
>ComPortReadOL))) {
                        // If the character was not read, try to figure out why:
                        DWORD error = GetLastError();
                        // If the IO operation is still pending, begin waiting until a character is read:
                        if (error == ERROR_IO_PENDING) {
                                while (! GetOverlappedResult (ComPort->hComPort, &(ComPort-
>ComPortReadOL), &BytesRead, FALSE)) {
                                        // If the thread should be terminated, abandon monitoring
comm port input:
                                        if (Terminated) {
                                                PurgeComm (ComPort->hComPort,
PURGE_RXABORT | PURGE_RXCLEAR);
                                                return;
                                        }
                                        // If the character was still not read, try to figure out why:
                                        error = GetLastError();
                                        // If the IO operation is still pending, wait for a maximum of
500 ms:
                                        if (error == ERROR_IO_INCOMPLETE)
                                                WaitForSingleObject (ComPort-
>hReadComPortDone, 5000);
                                        else
                                                // If there was another error, stop waiting for a
character:
                                                break;
                                }
                        }
                }
```

```
                // If a character was read after all this mess, process it:
                if (BytesRead == 1) {
                        ComPort->InputBuffer [ComPort->BytesRead] = LastByteRead;
                        Synchronize (ComPort->ReadByte);
                        if (ComPort->InputBuffer [ComPort->BytesRead]) {
                                ComPort->BytesRead ++;
                                ComPort->InputBuffer [ComPort->BytesRead] = 0;
                                if ((ComPort->InputBuffer [ComPort->BytesRead - 1] == ComPort-
>EndOfLineCharacter)
                                        || (ComPort->BytesRead == ComPort->OpenBufferSize - 1))
{
                                        Synchronize (ComPort->ReadLine);
                                        ComPort->BytesRead = 0;
                                }
                        }
                } else {
                        // If no character was read, an error occurred:
                        DWORD errors;
                        ClearCommError (ComPort->hComPort, &errors, NULL);
                }
        }
};
//---------------------------------------------------------------------------
__fastcall TWriteComPortLoop::TWriteComPortLoop(TComPort * Port)
        : TThread(true)
{
        ComPort = Port;
        FreeOnTerminate = true; // delete the object automatically when the thread terminates
        Resume ();
};
//---------------------------------------------------------------------------
void __fastcall TWriteComPortLoop::Execute()
{
        char LastByteWritten;
        DWORD BytesWritten;
        for (;;) {
                // If the thread should be terminated, abandon writing to the comm port:
                if (Terminated)
                        return;
                // If the port is not open, terminate the thread:
                if (ComPort->hComPort == INVALID_HANDLE_VALUE)
                        return;
                // If there is a byte to be written to the comm port, write it:
                if (ComPort->NextByteOut != ComPort->LastByteOut) {
                        // Allow the program to pre-process the byte to be written:
                        Synchronize (ComPort->WriteByte);
                        // Get the value of the next byte to be written:
                        LastByteWritten = ComPort->OutputBuffer [ComPort->NextByteOut];
                        ComPort->NextByteOut ++;
                        if (ComPort->NextByteOut == ComPort->OpenBufferSize)
                                ComPort->NextByteOut = 0;
                        if (LastByteWritten) {
                                // Try to write one character to the comm port:
```

```
                        if (! WriteFile (ComPort->hComPort, &LastByteWritten, 1,
&BytesWritten, &(ComPort->ComPortWriteOL))) {
                                // If the character was not written, try to figure out why:
                                DWORD error = GetLastError();
                                // If the IO operation is still pending, begin waiting until a
character is read:
                                if (error == ERROR_IO_PENDING) {
                                        while (! GetOverlappedResult (ComPort-
>hComPort, &(ComPort->ComPortWriteOL), &BytesWritten, FALSE)) {
                                                // If the thread should be terminated,
abandon writing to the comm port:
                                                if (Terminated) {
                                                        PurgeComm (ComPort-
>hComPort, PURGE_TXABORT | PURGE_TXCLEAR);
                                                        return;
                                                }
                                                // If the character was still not read, try to
figure out why:
                                                error = GetLastError();
                                                // If the IO operation is still pending, wait
for a maximum of 500 ms:
                                                if (error == ERROR_IO_INCOMPLETE)
                                                        WaitForSingleObject (ComPort-
>hWriteComPortDone, 5000);
                                                else
                                                        // If there was another error, stop
waiting for a character:
                                                        break;
                                        }
                                }
                        }
                        if (BytesWritten != 1) {
                                // If no character was written, an error occurred:
                                DWORD errors;
                                ClearCommError (ComPort->hComPort, &errors, NULL);
                        }
                }
        } else {
                // If there was no byte to write, wait until one becomes available:
                WaitForSingleObject (ComPort->hCharactersToWrite, 5000);
                ResetEvent (ComPort->hCharactersToWrite);
        }
    }
}
//---------------------------------------------------------------------------
```

_carp.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <math.h>
#pragma hdrstop
```

```
#include "_carp.h"
//---------------------------------------------------------------------------
#pragma resource "* dfm"
TCarp *Carp;
extern float sp_O2_min,sp_O2_max,sp_temp_min,sp_temp_max,sp_pH_min,sp_pH_max;
extern float sp_alk_min,sp_alk_max,sp_CO2_min,sp_CO2_max,sp_NH4_min,sp_NH4_max;
extern float sp_NO2_min,sp_NO2_max,sp_hard_min,sp_hard_max,sp_chl_min,sp_chl_max;
extern float sp_sal_min,sp_sal_max,sp_growth_factor,sp_b,sp_weight_fin;
extern float sp_start_wt,sp_market_weight;
extern int sp_start_day,sp_days_till_harvest;
extern int day;
extern int Tank_Number;
extern struct growth_data
{
   int  tank_num;       //tank number
   int  num_fish;       //number of fish in tank
        float sp_gr_fac;      //growth factor
   float sp_b;          //b factor
   float sp_fin_wt;     //final expected wt after 1 year
   float begin_wt;      //begining wt
   float present_wt;    //present weight
   float market_weight;  //market weight of fish
   float feed_rate;     //% body weight to feed at
   int  Start_Day;     //day culture was spawned
   int  Grow_Out_days;  //number of days until harvest size
   int  Days_Of_Culture;  //age of the culture
   char* sp_name;

}T[4];



//---------------------------------------------------------------------------
__fastcall TCarp::TCarp(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TCarp::BitBtn1Click(TObject *Sender)
{    //ok button sets all data and save
  sp_O2_min=atof(TCarp::low_DO->Text.c_str());
  sp_O2_max=atof(TCarp::high_DO->Text.c_str());
  sp_temp_min=atof(TCarp::low_temp->Text.c_str());
  sp_temp_max=atof(TCarp::high_temp->Text.c_str());
  sp_pH_min=atof(TCarp::low_temp->Text.c_str());
  sp_pH_max=atof(TCarp::high_temp->Text.c_str());
  sp_alk_min=atof(TCarp::low_temp->Text.c_str());
  sp_alk_max=atof(TCarp::high_temp->Text.c_str());
  sp_CO2_min=atof(TCarp::low_temp->Text.c_str());
  sp_CO2_max=atof(TCarp::high_temp->Text.c_str());
  sp_NH4_min=atof(TCarp::low_temp->Text.c_str());
  sp_NH4_max=atof(TCarp::high_temp->Text.c_str());
  sp_NO2_min=atof(TCarp::low_temp->Text.c_str());
```

```
sp_NO2_max=atof(TCarp::high_temp->Text.c_str());
sp_hard_min=atof(TCarp::low_temp->Text.c_str());
sp_hard_max=atof(TCarp::high_temp->Text.c_str());
sp_chl_min=atof(TCarp::low_temp->Text.c_str());
sp_chl_max=atof(TCarp::high_temp->Text.c_str());
sp_sal_min=atof(TCarp::low_temp->Text.c_str());
sp_sal_max=atof(TCarp::high_temp->Text.c_str());
sp_growth_factor=atof(TCarp::growth_factor->Text.c_str());
sp_b=atof(TCarp::b_factor->Text.c_str());
sp_weight_fin=atof(TCarp::weight_fin->Text.c_str());
sp_start_wt=atof(TCarp::S_Wt->Text.c_str());
sp_start_day=atoi(TCarp::S_day->Text.c_str());
sp_days_till_harvest=atoi(TCarp::D_U_H->Text.c_str());

 FILE *j;
switch (Tank_Number)
{
        case 0:
   j= fopen ("GROWTH[0].TXT", "w");
   if (j==NULL)return;
   break;
case 1:
   j= fopen ("GROWTH[1].TXT", "w");
   if (j==NULL)return;
   break;
case 2:
   j= fopen ("GROWTH[2].TXT", "w");
   if (j==NULL)return;
   break;
case 3:
   j= fopen ("GROWTH[3].TXT", "w");
   if (j==NULL)return;
   break;
}

   fprintf(j,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t%d\t%d\t%4.2f\t%d\t%c\t",
            T[Tank_Number].tank_num,T[Tank_Number].sp_gr_fac,T[Tank_Number].sp_b,
     T[Tank_Number].sp_fin_wt,T[Tank_Number].begin_wt,T[Tank_Number].present_wt,
     T[Tank_Number].feed_rate,T[Tank_Number].Start_Day,T[Tank_Number].Grow_Out_days,
     T[Tank_Number].Days_Of_Culture,T[Tank_Number].market_weight,
     T[Tank_Number].num_fish,T[Tank_Number].sp_name);
 fclose(j);

 Close();
}
//---------------------------------------------------------------------------
void __fastcall TCarp::FormActivate(TObject *Sender)
{
 Label22->Caption=IntToStr(Tank_Number+1);
 _spN->Caption=T[Tank_Number].sp_name;
 sp_growth_factor=atof(TCarp::growth_factor->Text.c_str());
 sp_b=atof(TCarp::b_factor->Text.c_str());
 sp_weight_fin=atof(TCarp::weight_fin->Text.c_str());
 sp_start_wt=atof(TCarp::S_Wt->Text.c_str());
```

```
  sp_start_day=atoi(TCarp::S_day->Text.c_str());
  sp_days_till_harvest=atoi(TCarp::D_U_H->Text.c_str());
  sp_market_weight=atof(TCarp::M_W->Text.c_str());
  if(T[Tank_Number].sp_gr_fac==0)T[Tank_Number].sp_gr_fac=sp_growth_factor;
  else growth_factor->Text=FloatToStr(T[Tank_Number].sp_gr_fac);
  if(T[Tank_Number].sp_b==0)T[Tank_Number].sp_b=sp_b;
  else b_factor->Text=FloatToStr(T[Tank_Number].sp_b);
  if(T[Tank_Number].sp_fin_wt==0)T[Tank_Number].sp_fin_wt=sp_weight_fin;
  else weight_fin->Text=FloatToStr(T[Tank_Number].sp_fin_wt);
  if(T[Tank_Number].begin_wt==0)T[Tank_Number].begin_wt=sp_start_wt;
  else S_Wt->Text=FloatToStr(T[Tank_Number].begin_wt);
  if(T[Tank_Number].Start_Day==0)T[Tank_Number].Start_Day=sp_start_day;
  else S_day->Text=FloatToStr(T[Tank_Number].Start_Day);
  if(T[Tank_Number].Grow_Out_days==0)T[Tank_Number].Grow_Out_days=sp_days_till_harvest;
  else D_U_H->Text=FloatToStr(T[Tank_Number].Grow_Out_days);
  if(T[Tank_Number].market_weight==0)T[Tank_Number].market_weight=sp_market_weight;
  else M_W->Text=FloatToStr(T[Tank_Number].market_weight);
  if(T[Tank_Number].Days_Of_Culture==0)
  {
   T[Tank_Number].Days_Of_Culture=(day-T[Tank_Number].Start_Day)+(int)(365*(1/-
T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
   D_O_C->Text=IntToStr(T[Tank_Number].Days_Of_Culture);
  }
  else D_O_C->Text=FloatToStr(T[Tank_Number].Days_Of_Culture);



}
//---------------------------------------------------------------------------
void __fastcall TCarp::S_WtChange(TObject *Sender)
{
  int wt_to_time;

  T[Tank_Number].begin_wt=atof(TCarp::S_Wt->Text.c_str());
  wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));

  T[Tank_Number].Days_Of_Culture=wt_to_time;
  D_O_C->Text=IntToStr(wt_to_time);
}
//---------------------------------------------------------------------------
void __fastcall TCarp::growth_factorChange(TObject *Sender)
{
        T[Tank_Number].sp_gr_fac=atof(TCarp::growth_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TCarp::weight_finChange(TObject *Sender)
{
 T[Tank_Number].sp_fin_wt=atof(TCarp::weight_fin->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TCarp::b_factorChange(TObject *Sender)
{
```

```
 T[Tank_Number].sp_b=atof(TCarp::b_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TCarp::S_dayChange(TObject *Sender)
{
 T[Tank_Number].Start_Day=atoi(TCarp::S_day->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TCarp::D_U_HChange(TObject *Sender)
{
 T[Tank_Number].Grow_Out_days=atof(TCarp::D_U_H->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TCarp::M_WChange(TObject *Sender)
{
        int wt_to_time;
   char *ErrorMessage="Reduce Market Weight";


  T[Tank_Number].market_weight=atof(TCarp::M_W->Text.c_str());
  if(T[Tank_Number].market_weight>=T[Tank_Number].sp_fin_wt)
    {
     Application->MessageBox(ErrorMessage, "Weight too high", MB_OK);
     M_W->Text="0";
     T[Tank_Number].market_weight=T[Tank_Number].sp_fin_wt-1;
    }
  if(T[Tank_Number].market_weight<T[Tank_Number].sp_fin_wt-1)
    {
        wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-pow((
T[Tank_Number].market_weight/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
                D_U_H->Text=wt_to_time-T[Tank_Number].Days_Of_Culture;
    }
}
//---------------------------------------------------------------------------
```

_catfish.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <math.h>
#pragma hdrstop

#include "_Catfish.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TCatfish *Catfish;
extern float sp_O2_min,sp_O2_max,sp_temp_min,sp_temp_max,sp_pH_min,sp_pH_max;
extern float sp_alk_min,sp_alk_max,sp_CO2_min,sp_CO2_max,sp_NH4_min,sp_NH4_max;
extern float sp_NO2_min,sp_NO2_max,sp_hard_min,sp_hard_max,sp_chl_min,sp_chl_max;
extern float sp_sal_min,sp_sal_max,sp_growth_factor,sp_b,sp_weight_fin;
extern float sp_start_wt,sp_market_weight;
extern int sp_start_day,sp_days_till_harvest;
extern int day;
```

```
extern int Tank_Number;
extern struct growth_data
{
   int  tank_num;       //tank number
   int  num_fish;       //number of fish in tank
        float sp_gr_fac;      //growth factor
   float sp_b;          //b factor
   float sp_fin_wt;     //final expected wt
   float begin_wt;      //begining wt
   float present_wt;    //present weight
   float market_weight;  //market weight of fish
   float feed_rate;     //% body wieght to feed at
   int  Start_Day;     //day culture was spawned
   int  Grow_Out_days;  //number of days until harvest size
   int  Days_Of_Culture;  //number of days the culture
   char *sp_name;

}T[4];
//---------------------------------------------------------------------------
__fastcall TCatfish::TCatfish(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TCatfish::BitBtn1Click(TObject *Sender)
{
  sp_O2_min=atof(TCatfish::low_DO->Text.c_str());
  sp_O2_max=atof(TCatfish::high_DO->Text.c_str());
  sp_temp_min=atof(TCatfish::low_temp->Text.c_str());
  sp_temp_max=atof(TCatfish::high_temp->Text.c_str());
  sp_pH_min=atof(TCatfish::low_temp->Text.c_str());
  sp_pH_max=atof(TCatfish::high_temp->Text.c_str());
  sp_alk_min=atof(TCatfish::low_temp->Text.c_str());
  sp_alk_max=atof(TCatfish::high_temp->Text.c_str());
  sp_CO2_min=atof(TCatfish::low_temp->Text.c_str());
  sp_CO2_max=atof(TCatfish::high_temp->Text.c_str());
  sp_NH4_min=atof(TCatfish::low_temp->Text.c_str());
  sp_NH4_max=atof(TCatfish::high_temp->Text.c_str());
  sp_NO2_min=atof(TCatfish::low_temp->Text.c_str());
  sp_NO2_max=atof(TCatfish::high_temp->Text.c_str());
  sp_hard_min=atof(TCatfish::low_temp->Text.c_str());
  sp_hard_max=atof(TCatfish::high_temp->Text.c_str());
  sp_chl_min=atof(TCatfish::low_temp->Text.c_str());
  sp_chl_max=atof(TCatfish::high_temp->Text.c_str());
  sp_sal_min=atof(TCatfish::low_temp->Text.c_str());
  sp_sal_max=atof(TCatfish::high_temp->Text.c_str());
  sp_growth_factor=atof(TCatfish::growth_factor->Text.c_str());
  sp_b=atof(TCatfish::b_factor->Text.c_str());
  sp_weight_fin=atof(TCatfish::weight_fin->Text.c_str());
  sp_start_wt=atof(TCatfish::S_Wt->Text.c_str());
  sp_start_day=atoi(TCatfish::S_day->Text.c_str());
  sp_days_till_harvest=atoi(TCatfish::D_U_H->Text.c_str());

        FILE *j;
```

```
switch (Tank_Number)
{
        case 0:
    j= fopen ("GROWTH[0].TXT", "w");
    if (j==NULL)return;
    break;
case 1:
    j= fopen ("GROWTH[1].TXT", "w");
    if (j==NULL)return;
    break;
case 2:
    j= fopen ("GROWTH[2].TXT", "w");
    if (j==NULL)return;
    break;
case 3:
    j= fopen ("GROWTH[3].TXT", "w");
    if (j==NULL)return;
    break;
}

 fprintf(j,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t%d\t%d\t%4.2f\t%d\t%s\t",
                T[Tank_Number].tank_num,T[Tank_Number].sp_gr_fac,T[Tank_Number].sp_b,
      T[Tank_Number].sp_fin_wt,T[Tank_Number].begin_wt,T[Tank_Number].present_wt,
      T[Tank_Number].feed_rate,T[Tank_Number].Start_Day,T[Tank_Number].Grow_Out_days,
      T[Tank_Number].Days_Of_Culture,T[Tank_Number].market_weight,
      T[Tank_Number].num_fish,T[Tank_Number].sp_name);
 fclose(j);

 Close();
}
//---------------------------------------------------------------------------
void __fastcall TCatfish::FormActivate(TObject *Sender)
{
        Label22->Caption=IntToStr(Tank_Number+1);
 _spN->Caption=T[Tank_Number].sp_name;
 sp_growth_factor=atof(TCatfish::growth_factor->Text.c_str());
 sp_b=atof(TCatfish::b_factor->Text.c_str());
 sp_weight_fin=atof(TCatfish::weight_fin->Text.c_str());
 sp_start_wt=atof(TCatfish::S_Wt->Text.c_str());
 sp_start_day=atoi(TCatfish::S_day->Text.c_str());
 sp_days_till_harvest=atoi(TCatfish::D_U_H->Text.c_str());
 sp_market_weight=atof(TCatfish::M_W->Text.c_str());
 if(T[Tank_Number].sp_gr_fac==0)T[Tank_Number].sp_gr_fac=sp_growth_factor;
 else growth_factor->Text=FloatToStr(T[Tank_Number].sp_gr_fac);
 if(T[Tank_Number].sp_b==0)T[Tank_Number].sp_b=sp_b;
 else b_factor->Text=FloatToStr(T[Tank_Number].sp_b);
 if(T[Tank_Number].sp_fin_wt==0)T[Tank_Number].sp_fin_wt=sp_weight_fin;
 else weight_fin->Text=FloatToStr(T[Tank_Number].sp_fin_wt);
 if(T[Tank_Number].begin_wt==0)T[Tank_Number].begin_wt=sp_start_wt;
 else S_Wt->Text=FloatToStr(T[Tank_Number].begin_wt);
 if(T[Tank_Number].Start_Day==0)T[Tank_Number].Start_Day=sp_start_day;
 else S_day->Text=FloatToStr(T[Tank_Number].Start_Day);
 if(T[Tank_Number].Grow_Out_days==0)T[Tank_Number].Grow_Out_days=sp_days_till_harvest;
 else D_U_H->Text=FloatToStr(T[Tank_Number].Grow_Out_days);
```

```
  if(T[Tank_Number].market_weight==0)T[Tank_Number].market_weight=sp_market_weight;
  else M_W->Text=FloatToStr(T[Tank_Number].market_weight);
  if(T[Tank_Number].Days_Of_Culture==0)
  {
   T[Tank_Number].Days_Of_Culture=(day-T[Tank_Number].Start_Day)+(int)(365*(1/-
T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
   D_O_C->Text=IntToStr(T[Tank_Number].Days_Of_Culture);
  }
  else D_O_C->Text=FloatToStr(T[Tank_Number].Days_Of_Culture);



}
//---------------------------------------------------------------------------
void __fastcall TCatfish::growth_factorChange(TObject *Sender)
{
        T[Tank_Number].sp_gr_fac=atof(TCatfish::growth_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TCatfish::S_WtChange(TObject *Sender)
{
        int wt_to_time;

   T[Tank_Number].begin_wt=atof(TCatfish::S_Wt->Text.c_str());
   wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));

   T[Tank_Number].Days_Of_Culture=wt_to_time;
   D_O_C->Text=IntToStr(wt_to_time);
}
//---------------------------------------------------------------------------
void __fastcall TCatfish::weight_finChange(TObject *Sender)
{
 T[Tank_Number].sp_fin_wt=atof(TCatfish::weight_fin->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TCatfish::b_factorChange(TObject *Sender)
{
 T[Tank_Number].sp_b=atof(TCatfish::b_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TCatfish::S_dayChange(TObject *Sender)
{
  T[Tank_Number].Start_Day=atoi(TCatfish::S_day->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TCatfish::D_U_HChange(TObject *Sender)
{
 T[Tank_Number].Grow_Out_days=atof(TCatfish::D_U_H->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TCatfish::M_WChange(TObject *Sender)
{
```

```
        int wt_to_time;
  char *ErrorMessage="Reduce Market Weight";


  T[Tank_Number].market_weight=atof(TCatfish::M_W->Text.c_str());
  if(T[Tank_Number].market_weight>=T[Tank_Number].sp_fin_wt)
   {
    Application->MessageBox(ErrorMessage, "Weight too high", MB_OK);
    M_W->Text="0";
    T[Tank_Number].market_weight=T[Tank_Number].sp_fin_wt-1;
   }
  if(T[Tank_Number].market_weight<T[Tank_Number].sp_fin_wt-1)
   {
        wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-pow((
T[Tank_Number].market_weight/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
                D_U_H->Text=wt_to_time-T[Tank_Number].Days_Of_Culture;
   }
}
//---------------------------------------------------------------------------
```

_Salmon.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <math.h>
#pragma hdrstop

#include "_Salmon.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TSalmon *Salmon;
extern float sp_O2_min,sp_O2_max,sp_temp_min,sp_temp_max,sp_pH_min,sp_pH_max;
extern float sp_alk_min,sp_alk_max,sp_CO2_min,sp_CO2_max,sp_NH4_min,sp_NH4_max;
extern float sp_NO2_min,sp_NO2_max,sp_hard_min,sp_hard_max,sp_chl_min,sp_chl_max;
extern float sp_sal_min,sp_sal_max,sp_growth_factor,sp_b,sp_weight_fin;
extern float sp_start_wt,sp_market_weight;
extern int sp_start_day,sp_days_till_harvest;
extern int day;
extern int Tank_Number;
extern struct growth_data
{
  int  tank_num;      //tank number
  int  num_fish;      //number of fish in tank
       float sp_gr_fac;     //growth factor
  float sp_b;          //b factor
  float sp_fin_wt;     //final expected wt
  float begin_wt;      //begining wt
  float present_wt;    //present weight
  float market_weight;  //market weight of fish
  float feed_rate;     //% body wieght to feed at
  int  Start_Day;     //day culture was spawned
  int  Grow_Out_days;  //number of days until harvest size
  int  Days_Of_Culture;  //number of days the culture
```

```
    char  *sp_name;

};T[4];
//---------------------------------------------------------------------------
__fastcall TSalmon::TSalmon(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TSalmon::BitBtn1Click(TObject *Sender)
{
 sp_O2_min=atof(TSalmon::low_DO->Text.c_str());
  sp_O2_max=atof(TSalmon::high_DO->Text.c_str());
  sp_temp_min=atof(TSalmon::low_temp->Text.c_str());
  sp_temp_max=atof(TSalmon::high_temp->Text.c_str());
  sp_pH_min=atof(TSalmon::low_temp->Text.c_str());
  sp_pH_max=atof(TSalmon::high_temp->Text.c_str());
  sp_alk_min=atof(TSalmon::low_temp->Text.c_str());
  sp_alk_max=atof(TSalmon::high_temp->Text.c_str());
  sp_CO2_min=atof(TSalmon::low_temp->Text.c_str());
  sp_CO2_max=atof(TSalmon::high_temp->Text.c_str());
  sp_NH4_min=atof(TSalmon::low_temp->Text.c_str());
  sp_NH4_max=atof(TSalmon::high_temp->Text.c_str());
  sp_NO2_min=atof(TSalmon::low_temp->Text.c_str());
  sp_NO2_max=atof(TSalmon::high_temp->Text.c_str());
  sp_hard_min=atof(TSalmon::low_temp->Text.c_str());
  sp_hard_max=atof(TSalmon::high_temp->Text.c_str());
  sp_chl_min=atof(TSalmon::low_temp->Text.c_str());
  sp_chl_max=atof(TSalmon::high_temp->Text.c_str());
  sp_sal_min=atof(TSalmon::low_temp->Text.c_str());
  sp_sal_max=atof(TSalmon::high_temp->Text.c_str());
  sp_growth_factor=atof(TSalmon::growth_factor->Text.c_str());
  sp_b=atof(TSalmon::b_factor->Text.c_str());
  sp_weight_fin=atof(TSalmon::weight_fin->Text.c_str());
  sp_start_wt=atof(TSalmon::S_Wt->Text.c_str());
  sp_start_day=atoi(TSalmon::S_day->Text.c_str());
  sp_days_till_harvest=atoi(TSalmon::D_U_H->Text.c_str());

        FILE *j;
 switch (Tank_Number)
 {
      case 0:
    j= fopen ("GROWTH[0].TXT", "w");
    if (j==NULL)return;
    break;
 case 1:
    j= fopen ("GROWTH[1].TXT", "w");
    if (j==NULL)return;
    break;
 case 2:
    j= fopen ("GROWTH[2].TXT", "w");
    if (j==NULL)return;
    break;
 case 3:
```

```
    j= fopen ("GROWTH[3].TXT", "w");
    if (j==NULL)return;
    break;
  }

 fprintf(j,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t%d\t%d\t%4.2f\t%d\t%s\t",
              T[Tank_Number].tank_num,T[Tank_Number].sp_gr_fac,T[Tank_Number].sp_b,
      T[Tank_Number].sp_fin_wt,T[Tank_Number].begin_wt,T[Tank_Number].present_wt,
      T[Tank_Number].feed_rate,T[Tank_Number].Start_Day,T[Tank_Number].Grow_Out_days,
      T[Tank_Number].Days_Of_Culture,T[Tank_Number].market_weight,
      T[Tank_Number].num_fish,T[Tank_Number].sp_name);
  fclose(j);

  Close();
}
//---------------------------------------------------------------------------
void __fastcall TSalmon::FormActivate(TObject *Sender)
{
        Label22->Caption=IntToStr(Tank_Number+1);
  _spN->Caption=T[Tank_Number].sp_name;
  sp_growth_factor=atof(TSalmon::growth_factor->Text.c_str());
  sp_b=atof(TSalmon::b_factor->Text.c_str());
  sp_weight_fin=atof(TSalmon::weight_fin->Text.c_str());
  sp_start_wt=atof(TSalmon::S_Wt->Text.c_str());
  sp_start_day=atoi(TSalmon::S_day->Text.c_str());
  sp_days_till_harvest=atoi(TSalmon::D_U_H->Text.c_str());
  sp_market_weight=atof(TSalmon::M_W->Text.c_str());
  if(T[Tank_Number].sp_gr_fac==0)T[Tank_Number].sp_gr_fac=sp_growth_factor;
  else growth_factor->Text=FloatToStr(T[Tank_Number].sp_gr_fac);
  if(T[Tank_Number].sp_b==0)T[Tank_Number].sp_b=sp_b;
  else b_factor->Text=FloatToStr(T[Tank_Number].sp_b);
  if(T[Tank_Number].sp_fin_wt==0)T[Tank_Number].sp_fin_wt=sp_weight_fin;
  else weight_fin->Text=FloatToStr(T[Tank_Number].sp_fin_wt);
  if(T[Tank_Number].begin_wt==0)T[Tank_Number].begin_wt=sp_start_wt;
  else S_Wt->Text=FloatToStr(T[Tank_Number].begin_wt);
  if(T[Tank_Number].Start_Day==0)T[Tank_Number].Start_Day=sp_start_day;
  else S_day->Text=FloatToStr(T[Tank_Number].Start_Day);
  if(T[Tank_Number].Grow_Out_days==0)T[Tank_Number].Grow_Out_days=sp_days_till_harvest;
  else D_U_H->Text=FloatToStr(T[Tank_Number].Grow_Out_days);
  if(T[Tank_Number].market_weight==0)T[Tank_Number].market_weight=sp_market_weight;
  else M_W->Text=FloatToStr(T[Tank_Number].market_weight);



}
//---------------------------------------------------------------------------
void __fastcall TSalmon::growth_factorChange(TObject *Sender)
{
        T[Tank_Number].sp_gr_fac=atof(TSalmon::growth_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TSalmon::S_WtChange(TObject *Sender)
{
        int wt_to_time;
```

```
   T[Tank_Number].begin_wt=atof(TSalmon::S_Wt->Text.c_str());
   wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));

   T[Tank_Number].Days_Of_Culture=wt_to_time;
   D_O_C->Text=IntToStr(wt_to_time);
}
//---------------------------------------------------------------------------
void __fastcall TSalmon::weight_finChange(TObject *Sender)
{
 T[Tank_Number].sp_fin_wt=atof(TSalmon::weight_fin->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TSalmon::b_factorChange(TObject *Sender)
{
  T[Tank_Number].sp_b=atof(TSalmon::b_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TSalmon::S_dayChange(TObject *Sender)
{
 T[Tank_Number].Start_Day=atoi(TSalmon::S_day->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TSalmon::D_U_HChange(TObject *Sender)
{
 T[Tank_Number].Grow_Out_days=atof(TSalmon::D_U_H->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TSalmon::M_WChange(TObject *Sender)
{
          int wt_to_time;
    char *ErrorMessage="Reduce Market Weight";


  T[Tank_Number].market_weight=atof(TSalmon::M_W->Text.c_str());
  if(T[Tank_Number].market_weight>=T[Tank_Number].sp_fin_wt)
    {
     Application->MessageBox(ErrorMessage, "Weight too high", MB_OK);
     M_W->Text="0";
     T[Tank_Number].market_weight=T[Tank_Number].sp_fin_wt-1;
    }
  if(T[Tank_Number].market_weight<T[Tank_Number].sp_fin_wt-1)
    {
          wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-pow((
T[Tank_Number].market_weight/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
                D_U_H->Text=wt_to_time-T[Tank_Number].Days_Of_Culture;
    }
}
//---------------------------------------------------------------------------
```

_Stripped_Bass.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <math.h>
#pragma hdrstop

#include "_Stripped_Bass.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TStripped_Bass *Stripped_Bass;
extern float sp_O2_min,sp_O2_max,sp_temp_min,sp_temp_max,sp_pH_min,sp_pH_max;
extern float sp_alk_min,sp_alk_max,sp_CO2_min,sp_CO2_max,sp_NH4_min,sp_NH4_max;
extern float sp_NO2_min,sp_NO2_max,sp_hard_min,sp_hard_max,sp_chl_min,sp_chl_max;
extern float sp_sal_min,sp_sal_max,sp_growth_factor,sp_b,sp_weight_fin;
extern float sp_start_wt,sp_market_weight;
extern int sp_start_day,sp_days_till_harvest;
extern int day;
extern int Tank_Number;
extern struct growth_data
{
   int   tank_num;       //tank number
   int   num_fish;       //number of fish in tank
         float sp_gr_fac;      //growth factor
   float sp_b;           //b factor
   float sp_fin_wt;      //final expected wt
   float begin_wt;       //begining wt
   float present_wt;     //present weight
   float market_weight;  //market weight of fish
   float feed_rate;      //% body wieght to feed at
   int   Start_Day;      //day culture was spawned
   int   Grow_Out_days;  //number of days until harvest size
   int   Days_Of_Culture;  //number of days the culture
   char  *sp_name;

}T[4];
//---------------------------------------------------------------------------
__fastcall TStripped_Bass::TStripped_Bass(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TStripped_Bass::BitBtn1Click(TObject *Sender)
{
 sp_O2_min=atof(TStripped_Bass::low_DO->Text.c_str());
  sp_O2_max=atof(TStripped_Bass::high_DO->Text.c_str());
  sp_temp_min=atof(TStripped_Bass::low_temp->Text.c_str());
  sp_temp_max=atof(TStripped_Bass::high_temp->Text.c_str());
  sp_pH_min=atof(TStripped_Bass::low_temp->Text.c_str());
  sp_pH_max=atof(TStripped_Bass::high_temp->Text.c_str());
  sp_alk_min=atof(TStripped_Bass::low_temp->Text.c_str());
  sp_alk_max=atof(TStripped_Bass::high_temp->Text.c_str());
  sp_CO2_min=atof(TStripped_Bass ::low_temp->Text.c_str());
```

```
sp_CO2_max=atof(TStripped_Bass::high_temp->Text.c_str());
sp_NH4_min=atof(TStripped_Bass::low_temp->Text.c_str());
sp_NH4_max=atof(TStripped_Bass::high_temp->Text.c_str());
sp_NO2_min=atof(TStripped_Bass::low_temp->Text.c_str());
sp_NO2_max=atof(TStripped_Bass::high_temp->Text.c_str());
sp_hard_min=atof(TStripped_Bass::low_temp->Text.c_str());
sp_hard_max=atof(TStripped_Bass::low_temp->Text.c_str());
sp_chl_max=atof(TStripped_Bass::high_temp->Text.c_str());
sp_sal_min=atof(TStripped_Bass::low_temp->Text.c_str());
sp_sal_max=atof(TStripped_Bass::high_temp->Text.c_str());
sp_growth_factor=atof(TStripped_Bass::growth_factor->Text.c_str());
sp_b=atof(TStripped_Bass::b_factor->Text.c_str());
sp_weight_fin=atof(TStripped_Bass::weight_fin->Text.c_str());
sp_start_wt=atof(TStripped_Bass::S_Wt->Text.c_str());
sp_start_day=atoi(TStripped_Bass::S_day->Text.c_str());
sp_days_till_harvest=atoi(TStripped_Bass::D_U_H->Text.c_str());

        FILE *j;
switch (Tank_Number)
{
        case 0:
   j= fopen ("GROWTH[0].TXT", "w");
   if (j==NULL)return;
   break;
case 1:
   j= fopen ("GROWTH[1].TXT", "w");
   if (j==NULL)return;
   break;
case 2:
   j= fopen ("GROWTH[2].TXT", "w");
   if (j==NULL)return;
   break;
case 3:
   j= fopen ("GROWTH[3].TXT", "w");
   if (j==NULL)return;
   break;
}

fprintf(j,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t%d\t%d\t%4.2f\t%d\t%s\t",
                T[Tank_Number].tank_num,T[Tank_Number].sp_gr_fac,T[Tank_Number].sp_b,
      T[Tank_Number].sp_fin_wt,T[Tank_Number].begin_wt,T[Tank_Number].present_wt,
      T[Tank_Number].feed_rate,T[Tank_Number].Start_Day,T[Tank_Number].Grow_Out_days,
      T[Tank_Number].Days_Of_Culture,T[Tank_Number].market_weight,
      T[Tank_Number].num_fish,T[Tank_Number].sp_name);
 fclose(j);

 Close();
}
//---------------------------------------------------------------------------
void __fastcall TStripped_Bass::FormActivate(TObject *Sender)
{
        Label22->Caption=IntToStr(Tank_Number+1);
 _spN->Caption=T[Tank_Number].sp_name;
 sp_growth_factor=atof(TStripped_Bass::growth_factor->Text.c_str());
```

```
  sp_b=atof(TStripped_Bass::b_factor->Text.c_str());
  sp_weight_fin=atof(TStripped_Bass::weight_fin->Text.c_str());
  sp_start_wt=atof(TStripped_Bass::S_Wt->Text.c_str());
  sp_start_day=atoi(TStripped_Bass::S_day->Text.c_str());
  sp_days_till_harvest=atoi(TStripped_Bass::D_U_H->Text.c_str());
  sp_market_weight=atof(TStripped_Bass::M_W->Text.c_str());
  if(T[Tank_Number].sp_gr_fac==0)T[Tank_Number].sp_gr_fac=sp_growth_factor;
  else growth_factor->Text=FloatToStr(T[Tank_Number].sp_gr_fac);
  if(T[Tank_Number].sp_b==0)T[Tank_Number].sp_b=sp_b;
  else b_factor->Text=FloatToStr(T[Tank_Number].sp_b);
  if(T[Tank_Number].sp_fin_wt==0)T[Tank_Number].sp_fin_wt=sp_weight_fin;
  else weight_fin->Text=FloatToStr(T[Tank_Number].sp_fin_wt);
  if(T[Tank_Number].begin_wt==0)T[Tank_Number].begin_wt=sp_start_wt;
  else S_Wt->Text=FloatToStr(T[Tank_Number].begin_wt);
  if(T[Tank_Number].Start_Day==0)T[Tank_Number].Start_Day=sp_start_day;
  else S_day->Text=FloatToStr(T[Tank_Number].Start_Day);
  if(T[Tank_Number].Grow_Out_days==0)T[Tank_Number].Grow_Out_days=sp_days_till_harvest;
  else D_U_H->Text=FloatToStr(T[Tank_Number].Grow_Out_days);
  if(T[Tank_Number].market_weight==0)T[Tank_Number].market_weight=sp_market_weight;
  else M_W->Text=FloatToStr(T[Tank_Number].market_weight);
  if(T[Tank_Number].Days_Of_Culture==0)
  {
   T[Tank_Number].Days_Of_Culture=(day-T[Tank_Number].Start_Day)+(int)(365*(1/-
T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
   D_O_C->Text=IntToStr(T[Tank_Number].Days_Of_Culture);
  }
  else D_O_C->Text=FloatToStr(T[Tank_Number].Days_Of_Culture);



}
//---------------------------------------------------------------------------
void __fastcall TStripped_Bass::growth_factorChange(TObject *Sender)
{
        T[Tank_Number].sp_gr_fac=atof(TStripped_Bass::growth_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TStripped_Bass::S_WtChange(TObject *Sender)
{
        int wt_to_time;

   T[Tank_Number].begin_wt=atof(TStripped_Bass::S_Wt->Text.c_str());
   wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));

   T[Tank_Number].Days_Of_Culture=wt_to_time;
   D_O_C->Text=IntToStr(wt_to_time);
}
//---------------------------------------------------------------------------
void __fastcall TStripped_Bass::weight_finChange(TObject *Sender)
{
 T[Tank_Number].sp_fin_wt=atof(TStripped_Bass::weight_fin->Text.c_str());
}
```

```
//---------------------------------------------------------------------------
void __fastcall TStripped_Bass::b_factorChange(TObject *Sender)
{
 T[Tank_Number].sp_b=atof(TStripped_Bass::b_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TStripped_Bass::S_dayChange(TObject *Sender)
{
 T[Tank_Number].Start_Day=atoi(TStripped_Bass::S_day->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TStripped_Bass::D_U_HChange(TObject *Sender)
{
 T[Tank_Number].Grow_Out_days=atof(TStripped_Bass::D_U_H->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TStripped_Bass::M_WChange(TObject *Sender)
{
        int wt_to_time;
   char *ErrorMessage="Reduce Markct Weight";


  T[Tank_Number].market_weight=atof(TStripped_Bass::M_W->Text.c_str());
  if(T[Tank_Number].market_weight>=T[Tank_Number].sp_fin_wt)
    {
     Application->MessageBox(ErrorMessage, "Weight too high", MB_OK);
     M_W->Text="0";
     T[Tank_Number].market_weight=T[Tank_Number].sp_fin_wt-1;
    }
  if(T[Tank_Number].market_weight<T[Tank_Number].sp_fin_wt-1)
    {
        wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-pow((
T[Tank_Number].market_weight/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
                D_U_H->Text=wt_to_time-T[Tank_Number].Days_Of_Culture;
    }
}
//---------------------------------------------------------------------------
```

_Sturgeon.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <math.h>
#pragma hdrstop

#include "_Sturgeon.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TSturgeon *Sturgeon;
 extern float sp_O2_min,sp_O2_max,sp_temp_min,sp_temp_max,sp_pH_min,sp_pH_max;
 extern float sp_alk_min,sp_alk_max,sp_CO2_min,sp_CO2_max,sp_NH4_min,sp_NH4_max;
 extern float sp_NO2_min,sp_NO2_max,sp_hard_min,sp_hard_max,sp_chl_min,sp_chl_max;
```

```
extern float sp_sal_min,sp_sal_max,sp_growth_factor,sp_b,sp_weight_fin;
extern float sp_start_wt,sp_market_weight;
extern int sp_start_day,sp_days_till_harvest;
extern int day;
extern int Tank_Number;
extern struct growth_data
{
   int  tank_num;       //tank number
   int  num_fish;       //number of fish in tank
        float sp_gr_fac;     //growth factor
   float sp_b;          //b factor
   float sp_fin_wt;     //final expected wt
   float begin_wt;      //begining wt
   float present_wt;    //present weight
   float market_weight;   //market weight of fish
   float feed_rate;     //% body wieght to feed at
   int  Start_Day;    //day culture was spawned
   int  Grow_Out_days;  //number of days until harvest size
   int  Days_Of_Culture;  //number of days the culture
   char *sp_name;

}T[4];
//---------------------------------------------------------------------------
__fastcall TSturgeon::TSturgeon(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TSturgeon::BitBtn1Click(TObject *Sender)
{
 sp_O2_min=atof(TSturgeon::low_DO->Text.c_str());
  sp_O2_max=atof(TSturgeon::high_DO->Text.c_str());
  sp_temp_min=atof(TSturgeon::low_temp->Text.c_str());
  sp_temp_max=atof(TSturgeon::high_temp->Text.c_str());
  sp_pH_min=atof(TSturgeon::low_pH->Text.c_str());
  sp_pH_max=atof(TSturgeon::high_pH->Text.c_str());
  sp_alk_min=atof(TSturgeon::low_alk->Text.c_str());
  sp_alk_max=atof(TSturgeon::high_alk->Text.c_str());
  sp_CO2_min=atof(TSturgeon::low_CO2->Text.c_str());
  sp_CO2_max=atof(TSturgeon::high_CO2->Text.c_str());
  sp_NH4_min=atof(TSturgeon::low_NH4->Text.c_str());
  sp_NH4_max=atof(TSturgeon::high_NH4->Text.c_str());
  sp_NO2_min=atof(TSturgeon::low_NO2->Text.c_str());
  sp_NO2_max=atof(TSturgeon::high_NO2->Text.c_str());
  sp_hard_min=atof(TSturgeon::low_hardness->Text.c_str());
  sp_hard_max=atof(TSturgeon::high_hardness->Text.c_str());
  sp_chl_min=atof(TSturgeon::low_chloride->Text.c_str());
  sp_chl_max=atof(TSturgeon::high_chloride->Text.c_str());
  sp_sal_min=atof(TSturgeon::low_salinity->Text.c_str());
  sp_sal_max=atof(TSturgeon::high_salinity->Text.c_str());
  sp_growth_factor=atof(TSturgeon::growth_factor->Text.c_str());
  sp_b=atof(TSturgeon::b_factor->Text.c_str());
  sp_weight_fin=atof(TSturgeon::weight_fin->Text.c_str());
  sp_start_wt=atof(TSturgeon::S_Wt->Text.c_str());
```

```
sp_start_day=atoi(TSturgeon::S_day->Text.c_str());
sp_days_till_harvest=atoi(TSturgeon::D_U_H->Text.c_str());

        FILE *j;
switch (Tank_Number)
{
      case 0:
   j= fopen ("GROWTH[0].TXT", "w");
   if (j==NULL)return;
   break;
case 1:
   j= fopen ("GROWTH[1].TXT", "w");
   if (j==NULL)return;
   break;
case 2:
   j= fopen ("GROWTH[2].TXT", "w");
   if (j==NULL)return;
   break;
case 3:
   j= fopen ("GROWTH[3].TXT", "w");
   if (j==NULL)return;
   break;
}

fprintf(j,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t%d\t%d\t%4.2f\t%d\t%s\t",
            T[Tank_Number].tank_num,T[Tank_Number].sp_gr_fac,T[Tank_Number].sp_b,
      T[Tank_Number].sp_fin_wt,T[Tank_Number].begin_wt,T[Tank_Number].present_wt,
      T[Tank_Number].feed_rate,T[Tank_Number].Start_Day,T[Tank_Number].Grow_Out_days,
      T[Tank_Number].Days_Of_Culture,T[Tank_Number].market_weight,
      T[Tank_Number].num_fish,T[Tank_Number].sp_name);
 fclose(j);

 Close();
}
//---------------------------------------------------------------------------
void __fastcall TSturgeon::FormActivate(TObject *Sender)
{
      Label22->Caption=IntToStr(Tank_Number+1);
 _spN->Caption=T[Tank_Number].sp_name;
 sp_growth_factor=atof(TSturgeon::growth_factor->Text.c_str());
 sp_b=atof(TSturgeon::b_factor->Text.c_str());
 sp_weight_fin=atof(TSturgeon::weight_fin->Text.c_str());
 sp_start_wt=atof(TSturgeon::S_Wt->Text.c_str());
 sp_start_day=atoi(TSturgeon::S_day->Text.c_str());
 sp_days_till_harvest=atoi(TSturgeon::D_U_H->Text.c_str());
 sp_market_weight=atof(TSturgeon::M_W->Text.c_str());
 if(T[Tank_Number].sp_gr_fac==0)T[Tank_Number].sp_gr_fac=sp_growth_factor;
 else growth_factor->Text=FloatToStr(T[Tank_Number].sp_gr_fac);
 if(T[Tank_Number].sp_b==0)T[Tank_Number].sp_b=sp_b;
 else b_factor->Text=FloatToStr(T[Tank_Number].sp_b);
 if(T[Tank_Number].sp_fin_wt==0)T[Tank_Number].sp_fin_wt=sp_weight_fin;
 else weight_fin->Text=FloatToStr(T[Tank_Number].sp_fin_wt);
 if(T[Tank_Number].begin_wt==0)T[Tank_Number].begin_wt=sp_start_wt;
 else S_Wt->Text=FloatToStr(T[Tank_Number].begin_wt);
```

```
  if(T[Tank_Number].Start_Day==0)T[Tank_Number].Start_Day=sp_start_day;
  else S_day->Text=FloatToStr(T[Tank_Number].Start_Day);
  if(T[Tank_Number].Grow_Out_days==0)T[Tank_Number].Grow_Out_days=sp_days_till_harvest;
  else D_U_H->Text=FloatToStr(T[Tank_Number].Grow_Out_days);
  if(T[Tank_Number].market_weight==0)T[Tank_Number].market_weight=sp_market_weight;
  else M_W->Text=FloatToStr(T[Tank_Number].market_weight);
  if(T[Tank_Number].Days_Of_Culture==0)
  {
   T[Tank_Number].Days_Of_Culture=(day-T[Tank_Number].Start_Day)+(int)(365*(1/-
T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
   D_O_C->Text=IntToStr(T[Tank_Number].Days_Of_Culture);
  }
  else D_O_C->Text=FloatToStr(T[Tank_Number].Days_Of_Culture);



}
//---------------------------------------------------------------------------
void __fastcall TSturgeon::growth_factorChange(TObject *Sender)
{
        T[Tank_Number].sp_gr_fac=atof(TSturgeon::growth_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TSturgeon::S_WtChange(TObject *Sender)
{
        int wt_to_time;

   T[Tank_Number].begin_wt=atof(TSturgeon::S_Wt->Text.c_str());
   wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));

   T[Tank_Number].Days_Of_Culture=wt_to_time;
   D_O_C->Text=IntToStr(wt_to_time);
}
//---------------------------------------------------------------------------
void __fastcall TSturgeon::weight_finChange(TObject *Sender)
{
 T[Tank_Number].sp_fin_wt=atof(TSturgeon::weight_fin->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TSturgeon::b_factorChange(TObject *Sender)
{
 T[Tank_Number].sp_b=atof(TSturgeon::b_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TSturgeon::S_dayChange(TObject *Sender)
{
 T[Tank_Number].Start_Day=atoi(TSturgeon::S_day->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TSturgeon::D_U_HChange(TObject *Sender)
{
 T[Tank_Number].Grow_Out_days=atof(TSturgeon::D_U_H->Text.c_str());
```

```
}
//---------------------------------------------------------------------------
void __fastcall TSturgeon::M_WChange(TObject *Sender)
{
        int wt_to_time;
   char *ErrorMessage="Reduce Market Weight";


  T[Tank_Number].market_weight=atof(TSturgeon::M_W->Text.c_str());
  if(T[Tank_Number].market_weight>=T[Tank_Number].sp_fin_wt)
    {
     Application->MessageBox(ErrorMessage, "Weight too high", MB_OK);
     M_W->Text="0";
     T[Tank_Number].market_weight=T[Tank_Number].sp_fin_wt-1;
    }
  if(T[Tank_Number].market_weight<T[Tank_Number].sp_fin_wt-1)
    {
        wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-pow((
T[Tank_Number].market_weight/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
                D_U_H->Text=wt_to_time-T[Tank_Number].Days_Of_Culture;
    }
}
//---------------------------------------------------------------------------
```

_Tilapia.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <math.h>
#pragma hdrstop

#include "_Tilapia.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TTilapia *Tilapia;
float sp_O2_min,sp_O2_max,sp_temp_min,sp_temp_max,sp_pH_min,sp_pH_max;
float sp_alk_min,sp_alk_max,sp_CO2_min,sp_CO2_max,sp_NH4_min,sp_NH4_max;
float sp_NO2_min,sp_NO2_max,sp_hard_min,sp_hard_max,sp_chl_min,sp_chl_max;
float sp_sal_min,sp_sal_max,sp_growth_factor=3.1,sp_b=14,sp_weight_fin=1363.0;
float sp_start_wt,sp_market_weight;
int sp_start_day,sp_days_till_harvest;
extern int day;
extern int Tank_Number;
extern struct growth_data
{
   int   tank_num;       //tank number
   int   num_fish;       //number of fish in tank
         float sp_gr_fac;      //growth factor
   float sp_b;           //b factor
   float sp_fin_wt;      //final expected wt after 1 year
```

```
  float begin_wt;       //begining wt
  float present_wt;     //present weight
  float market_weight;  //market weight of fish
  float feed_rate;      //% body wieght to feed at
  int  Start_Day_age;   //days old when culture began
  int  Start_Date;      //day culture began (0-365)
  int  Grow_Out_days;  //number of days until harvest size
  int  Days_Of_Culture;  //age of the culture
  char* sp_name;

}T[4];


//---------------------------------------------------------------------------
__fastcall TTilapia::TTilapia(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TTilapia::BitBtn1Click(TObject *Sender)  //close and save
{
  sp_O2_min=atof(TTilapia::low_DO->Text.c_str());
  sp_O2_max=atof(TTilapia::high_DO->Text.c_str());
  sp_temp_min=atof(TTilapia::low_temp->Text.c_str());
  sp_temp_max=atof(TTilapia::high_temp->Text.c_str());
  sp_pH_min=atof(TTilapia::low_pH->Text.c_str());
  sp_pH_max=atof(TTilapia::high_pH->Text.c_str());
  sp_alk_min=atof(TTilapia::low_alk->Text.c_str());
  sp_alk_max=atof(TTilapia::high_alk->Text.c_str());
  sp_CO2_min=atof(TTilapia::low_CO2->Text.c_str());
  sp_CO2_max=atof(TTilapia::high_CO2->Text.c_str());
  sp_NH4_min=atof(TTilapia::low_NH4->Text.c_str());
  sp_NH4_max=atof(TTilapia::high_NH4->Text.c_str());
  sp_NO2_min=atof(TTilapia::low_NO2->Text.c_str());
  sp_NO2_max=atof(TTilapia::high_NO2->Text.c_str());
  sp_hard_min=atof(TTilapia::low_hardness->Text.c_str());
  sp_hard_max=atof(TTilapia::high_hardness->Text.c_str());
  sp_chl_min=atof(TTilapia::low_chloride->Text.c_str());
  sp_chl_max=atof(TTilapia::high_chloride->Text.c_str());
  sp_sal_min=atof(TTilapia::low_salinity->Text.c_str());
  sp_sal_max=atof(TTilapia::high_salinity->Text.c_str());

   FILE *j;
  switch (Tank_Number)
  {
        case 0:
     j= fopen ("GROWTH[0].TXT", "w");
     if (j==NULL)return;
     break;
  case 1:
     j= fopen ("GROWTH[1].TXT", "w");
     if (j==NULL)return;
     break;
  case 2:
```

```
    j= fopen ("GROWTH[2].TXT", "w");
    if (j==NULL)return;
    break;
  case 3:
    j= fopen ("GROWTH[3].TXT", "w");
    if (j==NULL)return;
    break;
  }

  fprintf(j,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t%d\t%d\t%d\t%4.2f\t%d\t%c\t",
              T[Tank_Number].tank_num,T[Tank_Number].sp_gr_fac,T[Tank_Number].sp_b,
      T[Tank_Number].sp_fin_wt,T[Tank_Number].begin_wt,T[Tank_Number].present_wt,

T[Tank_Number].feed_rate,T[Tank_Number].Start_Day_age,T[Tank_Number].Start_Date,T[Tank_Num
ber].Grow_Out_days,
      T[Tank_Number].Days_Of_Culture,T[Tank_Number].market_weight,
      T[Tank_Number].num_fish,T[Tank_Number].sp_name);
  fclose(j);


  Close();
}
//---------------------------------------------------------------------------
void __fastcall TTilapia::FormActivate(TObject *Sender)
{
  int wt_to_time;
  int k;
  FILE *j;

  for(k=0;k<=3;k++)
  {
  switch (k)
  {
   case 0:
   j= fopen ("GROWTH[0].TXT", "r");
   if (j==NULL)return;
   break;
   case 1:
   j= fopen ("GROWTH[1].TXT", "r");
   if (j==NULL)return;
   break;
   case 2:
   j= fopen ("GROWTH[2].TXT", "r");
   if (j==NULL)return;
   break;
   case 3:
   j= fopen ("GROWTH[3].TXT", "r");
   if (j==NULL)return;
   break;
  }

  fscanf(j,"%d %f %f %f %f %f %d %d %d %f %d %c",
              &T[k].tank_num,&T[k].sp_gr_fac,&T[k].sp_b,&T[k].sp_fin_wt,
      &T[k].begin_wt,&T[k].present_wt,&T[k].feed_rate,
```

```
        &T[k].Start_Day_age,&T[k].Start_Date,&T[k].Grow_Out_days,
        &T[k].Days_Of_Culture,&T[k].market_weight,&T[k].num_fish,
        &T[k].sp_name);
  fclose(j);
  }

  Label22->Caption=IntToStr(Tank_Number+1);
//   _spN->Caption=T[Tank_Number].sp_name;

        //if data has been entrerd leave as is else update
  if(T[Tank_Number].sp_gr_fac==0)T[Tank_Number].sp_gr_fac=sp_growth_factor;
  else growth_factor->Text=FloatToStr(T[Tank_Number].sp_gr_fac);
  if(T[Tank_Number].sp_b==0)T[Tank_Number].sp_b=sp_b;
  else b_factor->Text=FloatToStr(T[Tank_Number].sp_b);
  if(T[Tank_Number].sp_fin_wt==0)T[Tank_Number].sp_fin_wt=sp_weight_fin;
  else weight_fin->Text=FloatToStr(T[Tank_Number].sp_fin_wt);
  if(T[Tank_Number].begin_wt==0)T[Tank_Number].begin_wt=sp_start_wt;
  else S_Wt->Text=FloatToStr(T[Tank_Number].begin_wt);
  if(T[Tank_Number].Start_Date==0)T[Tank_Number].Start_Date=sp_start_day;
  else S_day->Text=FloatToStr(T[Tank_Number].Start_Date);
  if(T[Tank_Number].Grow_Out_days==0)T[Tank_Number].Grow_Out_days=sp_days_till_harvest;
  else
  if(T[Tank_Number].market_weight==0)T[Tank_Number].market_weight=sp_market_weight;
  else
  M_W->Text=FloatToStr(T[Tank_Number].market_weight);
  if(T[Tank_Number].Start_Day_age==0)NULL;
  else
  D_A_S->Text=IntToStr(T[Tank_Number].Start_Day_age);

   T[Tank_Number].Days_Of_Culture=T[Tank_Number].Start_Day_age+(day-
T[Tank_Number].Start_Date);
   D_O_C->Text=IntToStr(T[Tank_Number].Days_Of_Culture);



              //time require for growout
  wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].market_weight/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
              //time until harverst
  T[Tank_Number].Grow_Out_days=wt_to_time-T[Tank_Number].Days_Of_Culture;
  D_U_H->Text=FloatToStr(T[Tank_Number].Grow_Out_days);
}
//---------------------------------------------------------------------------
void __fastcall TTilapia::S_WtChange(TObject *Sender)
{
  T[Tank_Number].begin_wt=atof(TTilapia::S_Wt->Text.c_str());
}
//---------------------------------------------------------------------------

void __fastcall TTilapia::D_U_HChange(TObject *Sender)
{
 T[Tank_Number].Grow_Out_days=atof(TTilapia::D_U_H->Text.c_str());
}
//---------------------------------------------------------------------------
```

```
void __fastcall TTilapia::weight_finChange(TObject *Sender)
{
 T[Tank_Number].sp_fin_wt=atof(TTilapia::weight_fin->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTilapia::b_factorChange(TObject *Sender)
{
 T[Tank_Number].sp_b=atof(TTilapia::b_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTilapia::S_dayChange(TObject *Sender)
{
 T[Tank_Number].Start_Date=atoi(TTilapia::S_day->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTilapia::growth_factorChange(TObject *Sender)
{
  T[Tank_Number].sp_gr_fac=atof(TTilapia::growth_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTilapia::D_A_SChange(TObject *Sender)
{
 T[Tank_Number].Start_Day_age=atoi(TTilapia::D_A_S->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTilapia::M_WChange(TObject *Sender)
{
        int wt_to_time;
   char *ErrorMessage="Reduce Market Weight";


  T[Tank_Number].market_weight=atof(TTilapia::M_W->Text.c_str());
  if(T[Tank_Number].market_weight>=T[Tank_Number].sp_fin_wt)
   {
    Application->MessageBox(ErrorMessage, "Weight too high", MB_OK);
    M_W->Text="0";
    T[Tank_Number].market_weight=T[Tank_Number].sp_fin_wt-1; //corrects to next lowest weight
   }
  if(T[Tank_Number].market_weight<T[Tank_Number].sp_fin_wt-1)
   {
     T[Tank_Number].Days_Of_Culture=T[Tank_Number].Start_Day_age+(day-
T[Tank_Number].Start_Date);
     D_O_C->Text=IntToStr(T[Tank_Number].Days_Of_Culture);
        wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-pow((
T[Tank_Number].market_weight/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
     Label25->Caption=IntToStr(wt_to_time);
                D_U_H->Text=wt_to_time-T[Tank_Number].Days_Of_Culture;
   }
}
//---------------------------------------------------------------------------
```

_Tropical.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <math.h>
#pragma hdrstop

#include "_Tropical.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TTropical *Tropical;
extern float sp_O2_min,sp_O2_max,sp_temp_min,sp_temp_max,sp_pH_min,sp_pH_max;
extern float sp_alk_min,sp_alk_max,sp_CO2_min,sp_CO2_max,sp_NH4_min,sp_NH4_max;
extern float sp_NO2_min,sp_NO2_max,sp_hard_min,sp_hard_max,sp_chl_min,sp_chl_max;
extern float sp_sal_min,sp_sal_max,sp_growth_factor,sp_b,sp_weight_fin;
extern float sp_start_wt,sp_market_weight;
extern int sp_start_day,sp_days_till_harvest;
extern int day;
extern int Tank_Number;
extern struct growth_data
{
   int   tank_num;        //tank number
   int   num_fish;        //number of fish in tank
         float sp_gr_fac;      //growth factor
   float sp_b;           //b factor
   float sp_fin_wt;       //final expected wt
   float begin_wt;        //begining wt
   float present_wt;      //present weight
   float market_weight;   //market weight of fish
   float feed_rate;       //% body wieght to feed at
   int   Start_Day;       //day culture was spawned
   int   Grow_Out_days;  //number of days until harvest size
   int   Days_Of_Culture;  //number of days the culture
   char  *sp_name;

}T[4];
//---------------------------------------------------------------------------
__fastcall TTropical::TTropical(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TTropical::BitBtn1Click(TObject *Sender)
{
  sp_O2_min=atof(TTropical::low_DO->Text.c_str());
  sp_O2_max=atof(TTropical::high_DO->Text.c_str());
  sp_temp_min=atof(TTropical::low_temp->Text.c_str());
  sp_temp_max=atof(TTropical::high_temp->Text.c_str());
  sp_pH_min=atof(TTropical::low_temp->Text.c_str());
  sp_pH_max=atof(TTropical::high_temp->Text.c_str());
  sp_alk_min=atof(TTropical::low_temp->Text.c_str());
  sp_alk_max=atof(TTropical::high_temp->Text.c_str());
  sp_CO2_min=atof(TTropical::low_temp->Text.c_str());
```

```
sp_CO2_max=atof(TTropical::high_temp->Text.c_str());
sp_NH4_min=atof(TTropical::low_temp->Text.c_str());
sp_NH4_max=atof(TTropical::high_temp->Text.c_str());
sp_NO2_min=atof(TTropical::low_temp->Text.c_str());
sp_NO2_max=atof(TTropical::high_temp->Text.c_str());
sp_hard_min=atof(TTropical::low_temp->Text.c_str());
sp_hard_max=atof(TTropical::high_temp->Text.c_str());
sp_chl_min=atof(TTropical::low_temp->Text.c_str());
sp_chl_max=atof(TTropical::high_temp->Text.c_str());
sp_sal_min=atof(TTropical::low_temp->Text.c_str());
sp_sal_max=atof(TTropical::high_temp->Text.c_str());
sp_growth_factor=atof(TTropical::growth_factor->Text.c_str());
sp_b=atof(TTropical::b_factor->Text.c_str());
sp_weight_fin=atof(TTropical::weight_fin->Text.c_str());
sp_start_wt=atof(TTropical::S_Wt->Text.c_str());
sp_start_day=atoi(TTropical::S_day->Text.c_str());
sp_days_till_harvest=atoi(TTropical::D_U_H->Text.c_str());

        FILE *j;
switch (Tank_Number)
{
      case 0:
   j= fopen ("GROWTH[0].TXT", "w");
   if (j==NULL)return;
   break;
 case 1:
   j= fopen ("GROWTH[1].TXT", "w");
   if (j==NULL)return;
   break;
 case 2:
   j= fopen ("GROWTH[2].TXT", "w");
   if (j==NULL)return;
   break;
 case 3:
   j= fopen ("GROWTH[3].TXT", "w");
   if (j==NULL)return;
   break;
 }

 fprintf(j,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t%d\t%d\t%4.2f\t%d\t%s\t",
                T[Tank_Number].tank_num,T[Tank_Number].sp_gr_fac,T[Tank_Number].sp_b,
      T[Tank_Number].sp_fin_wt,T[Tank_Number].begin_wt,T[Tank_Number].present_wt,
      T[Tank_Number].feed_rate,T[Tank_Number].Start_Day,T[Tank_Number].Grow_Out_days,
      T[Tank_Number].Days_Of_Culture,T[Tank_Number].market_weight,
      T[Tank_Number].num_fish,T[Tank_Number].sp_name);
 fclose(j);

 Close();
}
//---------------------------------------------------------------------------
void __fastcall TTropical::FormActivate(TObject *Sender)
{
      Label22->Caption=IntToStr(Tank_Number+1);
 _spN->Caption=T[Tank_Number].sp_name;
```

```
  sp_growth_factor=atof(TTropical::growth_factor->Text.c_str());
  sp_b=atof(TTropical::b_factor->Text.c_str());
  sp_weight_fin=atof(TTropical::weight_fin->Text.c_str());
  sp_start_wt=atof(TTropical::S_Wt->Text.c_str());
  sp_start_day=atoi(TTropical::S_day->Text.c_str());
  sp_days_till_harvest=atoi(TTropical::D_U_H->Text.c_str());
  sp_market_weight=atof(TTropical::M_W->Text.c_str());
  if(T[Tank_Number].sp_gr_fac==0)T[Tank_Number].sp_gr_fac=sp_growth_factor;
  else growth_factor->Text=FloatToStr(T[Tank_Number].sp_gr_fac);
  if(T[Tank_Number].sp_b==0)T[Tank_Number].sp_b=sp_b;
  else b_factor->Text=FloatToStr(T[Tank_Number].sp_b);
  if(T[Tank_Number].sp_fin_wt==0)T[Tank_Number].sp_fin_wt=sp_weight_fin;
  else weight_fin->Text=FloatToStr(T[Tank_Number].sp_fin_wt);
  if(T[Tank_Number].begin_wt==0)T[Tank_Number].begin_wt=sp_start_wt;
  else S_Wt->Text=FloatToStr(T[Tank_Number].begin_wt);
  if(T[Tank_Number].Start_Day==0)T[Tank_Number].Start_Day=sp_start_day;
  else S_day->Text=FloatToStr(T[Tank_Number].Start_Day);
  if(T[Tank_Number].Grow_Out_days==0)T[Tank_Number].Grow_Out_days=sp_days_till_harvest;
  else D_U_H->Text=FloatToStr(T[Tank_Number].Grow_Out_days);
  if(T[Tank_Number].market_weight==0)T[Tank_Number].market_weight=sp_market_weight;
  else M_W->Text=FloatToStr(T[Tank_Number].market_weight);
  if(T[Tank_Number].Days_Of_Culture==0)
  {
   T[Tank_Number].Days_Of_Culture=(day-T[Tank_Number].Start_Day)+(int)(365*(1/-
T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
   D_O_C->Text=IntToStr(T[Tank_Number].Days_Of_Culture);
  }
  else D_O_C->Text=FloatToStr(T[Tank_Number].Days_Of_Culture);



}
//---------------------------------------------------------------------------
void __fastcall TTropical::growth_factorChange(TObject *Sender)
{
        T[Tank_Number].sp_gr_fac=atof(TTropical::growth_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTropical::S_WtChange(TObject *Sender)
{
         int wt_to_time;

  T[Tank_Number].begin_wt=atof(TTropical::S_Wt->Text.c_str());
  wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));

  T[Tank_Number].Days_Of_Culture=wt_to_time;
  D_O_C->Text=IntToStr(wt_to_time);
}
//---------------------------------------------------------------------------
void __fastcall TTropical::weight_finChange(TObject *Sender)
{
 T[Tank_Number].sp_fin_wt=atof(TTropical::weight_fin->Text.c_str());
```

```
}
//---------------------------------------------------------------------------
void __fastcall TTropical::b_factorChange(TObject *Sender)
{
 T[Tank_Number].sp_b=atof(TTropical::b_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTropical::S_dayChange(TObject *Sender)
{
 T[Tank_Number].Start_Day=atoi(TTropical::S_day->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTropical::D_U_HChange(TObject *Sender)
{
 T[Tank_Number].Grow_Out_days=atof(TTropical::D_U_H->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTropical::M_WChange(TObject *Sender)
{
        int wt_to_time;
   char *ErrorMessage="Reduce Market Weight";


  T[Tank_Number].market_weight=atof(TTropical::M_W->Text.c_str());
  if(T[Tank_Number].market_weight>=T[Tank_Number].sp_fin_wt)
   {
    Application->MessageBox(ErrorMessage, "Weight too high", MB_OK);
    M_W->Text="0";
    T[Tank_Number].market_weight=T[Tank_Number].sp_fin_wt-1;
   }
  if(T[Tank_Number].market_weight<T[Tank_Number].sp_fin_wt-1)
   {
       wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-pow((
T[Tank_Number].market_weight/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
                D_U_H->Text=wt_to_time-T[Tank_Number].Days_Of_Culture;
   }
}
//---------------------------------------------------------------------------
```

_Trout.cpp

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#include <math.h>
#pragma hdrstop

#include "_Trout.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TTrout *Trout;
extern float sp_O2_min,sp_O2_max,sp_temp_min,sp_temp_max,sp_pH_min,sp_pH_max;
extern float sp_alk_min,sp_alk_max,sp_CO2_min,sp_CO2_max,sp_NH4_min,sp_NH4_max;
extern float sp_NO2_min,sp_NO2_max,sp_hard_min,sp_hard_max,sp_chl_min,sp_chl_max;
```

```
extern float sp_sal_min,sp_sal_max,sp_growth_factor,sp_b,sp_weight_fin;
extern float sp_start_wt,sp_market_weight;
extern int sp_start_day,sp_days_till_harvest;
extern int day;
extern int Tank_Number;
extern struct growth_data
{
   int   tank_num;       //tank number
   int   num_fish;       //number of fish in tank
         float sp_gr_fac;      //growth factor
   float sp_b;           //b factor
   float sp_fin_wt;      //final expected wt
   float begin_wt;       //begining wt
   float present_wt;     //present weight
   float market_weight;  //market weight of fish
   float feed_rate;      //% body wieght to feed at
   int   Start_Day;      //day culture was spawned
   int   Grow_Out_days;  //number of days until harvest size
   int   Days_Of_Culture;  //number of days the culture
   char  *sp_name;

}T[4];
//---------------------------------------------------------------------------
__fastcall TTrout::TTrout(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TTrout::OKClick(TObject *Sender)
{
  sp_O2_min=atof(TTrout::low_DO->Text.c_str());
  sp_O2_max=atof(TTrout::high_DO->Text.c_str());
  sp_temp_min=atof(TTrout::low_temp->Text.c_str());
  sp_temp_max=atof(TTrout::high_temp->Text.c_str());
  sp_pH_min=atof(TTrout::low_temp->Text.c_str());
  sp_pH_max=atof(TTrout::high_temp->Text.c_str());
  sp_alk_min=atof(TTrout::low_temp->Text.c_str());
  sp_alk_max=atof(TTrout::high_temp->Text.c_str());
  sp_CO2_min=atof(TTrout::low_temp->Text.c_str());
  sp_CO2_max=atof(TTrout::high_temp->Text.c_str());
  sp_NH4_min=atof(TTrout::low_temp->Text.c_str());
  sp_NH4_max=atof(TTrout::high_temp->Text.c_str());
  sp_NO2_min=atof(TTrout::low_temp->Text.c_str());
  sp_NO2_max=atof(TTrout::high_temp->Text.c_str());
  sp_hard_min=atof(TTrout::low_temp->Text.c_str());
  sp_hard_max=atof(TTrout::high_temp->Text.c_str());
  sp_chl_min=atof(TTrout::low_temp->Text.c_str());
  sp_chl_max=atof(TTrout::high_temp->Text.c_str());
  sp_sal_min=atof(TTrout::low_temp->Text.c_str());
  sp_sal_max=atof(TTrout::high_temp->Text.c_str());

        FILE *j;
  switch (Tank_Number)
  {
```

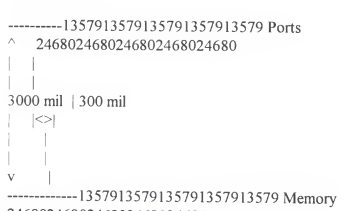
```
        case 0:
      j= fopen ("GROWTH[0].TXT", "w");
      if (j==NULL)return;
      break;
    case 1:
      j= fopen ("GROWTH[1].TXT", "w");
      if (j==NULL)return;
      break;
    case 2:
      j= fopen ("GROWTH[2].TXT", "w");
      if (j==NULL)return;
      break;
    case 3:
      j= fopen ("GROWTH[3].TXT", "w");
      if (j==NULL)return;
      break;
    }

  fprintf(j,"%d\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%d\t%d\t%d\t%4.2f\t%d\t%s\t",
                T[Tank_Number].tank_num,T[Tank_Number].sp_gr_fac,T[Tank_Number].sp_b,
        T[Tank_Number].sp_fin_wt,T[Tank_Number].begin_wt,T[Tank_Number].present_wt,
        T[Tank_Number].feed_rate,T[Tank_Number].Start_Day,T[Tank_Number].Grow_Out_days,
        T[Tank_Number].Days_Of_Culture,T[Tank_Number].market_weight,
        T[Tank_Number].num_fish,T[Tank_Number].sp_name);
    fclose(j);

    Close();
}
//---------------------------------------------------------------------------
void __fastcall TTrout::FormActivate(TObject *Sender)
{
          Label22->Caption=IntToStr(Tank_Number+1);
  _spN->Caption=T[Tank_Number].sp_name;
  sp_growth_factor=atof(TTrout::growth_factor->Text.c_str());
  sp_b=atof(TTrout::b_factor->Text.c_str());
  sp_weight_fin=atof(TTrout::weight_fin->Text.c_str());
  sp_start_wt=atof(TTrout::S_Wt->Text.c_str());
  sp_start_day=atoi(TTrout::S_day->Text.c_str());
  sp_days_till_harvest=atoi(TTrout::D_U_H->Text.c_str());
  sp_market_weight=atof(TTrout::M_W->Text.c_str());
  if(T[Tank_Number].sp_gr_fac==0)T[Tank_Number].sp_gr_fac=sp_growth_factor;
  else growth_factor->Text=FloatToStr(T[Tank_Number].sp_gr_fac);
  if(T[Tank_Number].sp_b==0)T[Tank_Number].sp_b=sp_b;
  else b_factor->Text=FloatToStr(T[Tank_Number].sp_b);
  if(T[Tank_Number].sp_fin_wt==0)T[Tank_Number].sp_fin_wt=sp_weight_fin;
  else weight_fin->Text=FloatToStr(T[Tank_Number].sp_fin_wt);
  if(T[Tank_Number].begin_wt==0)T[Tank_Number].begin_wt=sp_start_wt;
  else S_Wt->Text=FloatToStr(T[Tank_Number].begin_wt);
  if(T[Tank_Number].Start_Day==0)T[Tank_Number].Start_Day=sp_start_day;
  else S_day->Text=FloatToStr(T[Tank_Number].Start_Day);
  if(T[Tank_Number].Grow_Out_days==0)T[Tank_Number].Grow_Out_days=sp_days_till_harvest;
  else D_U_H->Text=FloatToStr(T[Tank_Number].Grow_Out_days);
  if(T[Tank_Number].market_weight==0)T[Tank_Number].market_weight=sp_market_weight;
  else M_W->Text=FloatToStr(T[Tank_Number].market_weight);
```

```
  if(T[Tank_Number].Days_Of_Culture==0)
  {
   T[Tank_Number].Days_Of_Culture=(day-T[Tank_Number].Start_Day)+(int)(365*(1/-
T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
   D_O_C->Text=IntToStr(T[Tank_Number].Days_Of_Culture);
  }
  else D_O_C->Text=FloatToStr(T[Tank_Number].Days_Of_Culture);



}
//---------------------------------------------------------------------------
void __fastcall TTrout::growth_factorChange(TObject *Sender)
{
        T[Tank_Number].sp_gr_fac=atof(TTrout::growth_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTrout::S_WtChange(TObject *Sender)
{
          int wt_to_time;

   T[Tank_Number].begin_wt=atof(TTrout::S_Wt->Text.c_str());
   wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-
pow((T[Tank_Number].begin_wt/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));

   T[Tank_Number].Days_Of_Culture=wt_to_time;
   D_O_C->Text=IntToStr(wt_to_time);
}
//---------------------------------------------------------------------------
void __fastcall TTrout::weight_finChange(TObject *Sender)
{
 T[Tank_Number].sp_fin_wt=atof(TTrout::weight_fin->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTrout::b_factorChange(TObject *Sender)
{
 T[Tank_Number].sp_b=atof(TTrout::b_factor->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTrout::S_dayChange(TObject *Sender)
{
 T[Tank_Number].Start_Day=atoi(TTrout::S_day->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTrout::D_U_HChange(TObject *Sender)
{
 T[Tank_Number].Grow_Out_days=atof(TTrout::D_U_H->Text.c_str());
}
//---------------------------------------------------------------------------
void __fastcall TTrout::M_WChange(TObject *Sender)
{
          int wt_to_time;
   char *ErrorMessage="Reduce Market Weight";
```

```
  T[Tank_Number].market_weight=atof(TTrout::M_W->Text.c_str());
  if(T[Tank_Number].market_weight>=T[Tank_Number].sp_fin_wt)
    {
     Application->MessageBox(ErrorMessage, "Weight too high", MB_OK);
     M_W->Text="0";
     T[Tank_Number].market_weight=T[Tank_Number].sp_fin_wt-1;
    }
  if(T[Tank_Number].market_weight<T[Tank_Number].sp_fin_wt-1)
    {
          wt_to_time=(int)(365*(1/-T[Tank_Number].sp_gr_fac)*log(1-pow((
T[Tank_Number].market_weight/T[Tank_Number].sp_fin_wt),1/T[Tank_Number].sp_b)));
                    D_U_H->Text=wt_to_time-T[Tank_Number].Days_Of_Culture;
    }
}
//---------------------------------------------------------------------------
```

APPENDIX E
PINOUT AND CONNECTIONS

Header layout.



The memory and ports headers.

```
      Memory                          Ports
      -------                         -------
GND  ---| 1  2|--- Vcc          GND   ---| 1  2|--- Vcc
A15  ---| 3  4|--- A14          CANH  ---| 3  4|--- CANL

A12  ---| 5  6|--- A13          RST   ---| 5  6|--- STADC
A7   ---| 7  8|--- A8           Vref- ---| 7  8|--- Vref+
A6   ---| 9 10|--- A9           P1.0(RxD) ---| 9 10|--- P1.1(TxD)
A5   ---|11 12|--- A11          P1.2(I0)  ---|11 12|--- P1.3(I1)
A4   ---|13 14|--- PSEN         P1.4(T0)  ---|13 14|--- P1.5(T1)
A3   ---|15 16|--- A10          P1.6(I2)  ---|15 16|--- P1.7(I3)
A2   ---|17 18|--- RD           P2.0(Bus) ---|17 18|--- P2.1(Bus)
A1   ---|19 20|--- D7           P2.2(Bus) ---|19 20|--- P2.3(Bus)
A0   ---|21 22|--- D6           P2.4(Bus) ---|21 22|--- P2.5(Bus)
D0   ---|23 24|--- D5           P2.6(Bus) ---|23 24|--- P2.7(Bus)
D1   ---|25 26|--- D4           P3.0(Adc) ---|25 26|--- P3.1(Adc)
D2   ---|27 28|--- D3           P3.2(Adc) ---|27 28|--- P3.3(Adc)
WR   ---|29 30|--- ALE          P3.4(Adc) ---|29 30|--- P3.5(Adc)
CTX0 ---|31 32|--- CRX0         P3.6(Adc) ---|31 32|--- P3.7(Adc)
CTX1 ---|33 34|--- CRX1         P4.0(Pwm) ---|33 34|--- P4.1(Pwm)
      -------       -------
A16  ---|35 36|--- A17          P4.2  ---|35 36|--- P4.3
A18  ---|37 38|--- A19          P4.4  ---|37 38|--- P4.5
A20  ---|39 40|--- A21          P4.6  ---|39 40|--- P4.7
      -------       -------
A22  ---|41 42|--- A23          P5.0 ---|41 42|--- P5.1
D8   ---|43 44|--- D9                P5.2 ---|43 44|--- P5.3
D10  ---|45 46|--- D11          P5.4 ---|45 46|--- P5.5
D12  ---|47 48|--- D13          P5.6 ---|47 48|--- P5.7
D14  ---|49 50|--- D15                ---|49 50|---
      -------       -------
```

CAN-592 motherboard

The motherboard for the Philips P80C592 has a 34-pin header for memory and a 40-pin header for ports. The connections are as follows:

```
      Memory                                  Ports
      -------                                 -------
GND ---| 1  2|--- Vcc                 GND     ---| 1  2|--- Vcc
P2.7 ---| 3  4|--- P2.6               CANH    ---| 3  4|--- CANL
P2.4 ---| 5  6|--- P2.5               RST     ---| 5  6|--- STADC
A7*  ---| 7  8|--- P2.0               AVref-  ---| 7  8|--- AVref+
A6*  ---| 9 10|--- P2.1               P3.0(RxD) ---| 9 10|--- P3.1(TxD)
A5*  ---|11 12|--- P2.3               P3.2(I0)  ---|11 12|--- P3.3(I1)
A4*  ---|13 14|--- PSEN              P3.4(T0)  ---|13 14|--- P3.5(T1)
A3*  ---|15 16|--- P2.2               P1.0(I2)  ---|15 16|--- P1.1(I3)
A2*  ---|17 18|--- P3.7               P4.0(Bus) ---|17 18|--- P4.1(Bus)
A1*  ---|19 20|--- P0.7               P4.2(Bus) ---|19 20|--- P4.3(Bus)
A0*  ---|21 22|--- P0.6               P4.4(Bus) ---|21 22|--- P4.5(Bus)
P0.0 ---|23 24|--- P0.5               P4.6(Bus) ---|23 24|--- P4.7(Bus)
P0.1 ---|25 26|--- P0.4               P5.0(Adc) ---|25 26|--- P5.1(Adc)
P0.2 ---|27 28|--- P0.3               P5.2(Adc) ---|27 28|--- P5.3(Adc)
P3.6 ---|29 30|--- ALE                P5.4(Adc) ---|29 30|--- P5.5(Adc)
P1.6 ---|31 32|--- CRX0               P5.6(Adc) ---|31 32|--- P5.7(Adc)
P1.7 ---|33 34|--- CRX1               PWM0    ---|33 34|--- PWM1
      -------                 -------
P1.2 ---|35 36|--- P1.3

P1.4 ---|37 38|--- P1.5
     ---|39 40|---
```

*A0-A7 are connected to P0.0-P0.7 via the 74HC573 latch.

LCD pinout.

```
      -------
Vcc  ---| 2  1|--- GND
RS   ---| 4  3|--- Vee
E    ---| 6  5|--- R/W
DB1  ---| 8  7|--- DB0
DB3  ---|10  9|--- DB2
DB5  ---|12 11|--- DB4
DB7  ---|14 13|--- DB6
      -------
```

RJ-11 Connector Cable

```
1  /-------------\============ + DC power
2  | /------\    |============ CANH
3  | |       -------\========== GND
4  | |       -------/========== GND
5  | \------/    |============ CANL
6  \-------------/============ - DC power
```

CLINK header pinout.

Memory

```
       -------
 GND ---| 1  2|--- Vcc
~CE# ---| 3  4|--- A14
 A12 ---| 5  6|--- A13
 A7  ---| 7  8|--- A8
 A6  ---| 9 10|--- A9
 A5  ---|11 12|--- A11
 A4  ---|13 14|---
 A3  ---|15 16|--- A10
 A2  ---|17 18|--- OE#
 A1  ---|19 20|--- D7
 A0  ---|21 22|--- D6
 D0  ---|23 24|--- D5
 D1  ---|25 26|--- D4
 D2  ---|27 28|--- D3
 WE# ---|29 30|---
        ---|31 32|---
        ---|33 34|---
       -------
```

Ports

```
       -------
GND          ---| 1  2|--- Vcc
RJ11.CANH    ---| 3  4|--- RJ11.CANL
 ResetBtn    ---| 5  6|---
             ---| 7  8|---
RS232.RxD    ---| 9 10|--- RS232.TxD
A15/<AS0#>   ---|11 12|--- A16<AS1#>
A17/<RD#>    ---|13 14|--- A18<WR#>
[E/E1]/Red   ---|15 16|--- [E2]/Yellow
[DB0]/<AD0>  ---|17 18|--- [DB1]/<AD1>
[DB2]/<AD2>  ---|19 20|--- [DB3]/<AD3>
[DB4]/<AD4>  ---|21 22|--- [DB5]/<AD5>
[DB6]/<AD6>  ---|23 24|--- [DB7]/<AD7>
Button1      ---|25 26|--- Button2
Button3      ---|27 28|--- Button4
Switch       ---|29 30|--- Jumper1
      <SQW>  ---|31 32|--- <IRQ#>
PwrSwitch1   ---|33 34|--- PwrSwitch2
       -------
```

```
                [RS]/<ALE>   ---|35 36|--- [RW]/~<CS#>
                <OER#/MOT>   ---|37 38|---
<WER#/RST#>
                                ---|39 40|---
                       -------
```

Notes:

1. The Memory header connections are to the AMD Am29F040B Flash memory chip.
2. Connections to the LCD are marked by [ ]; connections to the RTC are marked by < >.
2. Memory pin 3 is connected to the CE# (chip enable) via a logical NOT.
3. Pins 3-4 on the Ports header are connected to the RJ-11 containing the CAN/power connections.
4. Pin 5 on the Ports header is connected to a Reset pushbutton.
5. Pins 9-10 on the Ports header are connected to a MAX232 chip.
6. Pins 11-14 on the Ports header
3. Pins 17-24 on the Ports header are for an 8-bit bus that is shared between the LCD data lines and the RTC address/data lines. The lines will be used by the device which is enabled at the time.

Agua header pinout
Ports

```
          -------
GND       ---| 1  2|--- Vcc
RJ11.CANH ---| 3  4|--- RJ11.CANL
ResetBtn  ---| 5  6|---
          ---| 7  8|---
RS232.RxD ---| 9 10|--- RS232.TxD
<IRQ#>    ---|11 12|--- <1Hz>
Pulse     ---|13 14|--- OneWireBus
[E/E1]/Red---|15 16|--- [E2]/Yellow
[DB0]     ---|17 18|--- [DB1]
[DB2]     ---|19 20|--- [DB3]
[DB4]     ---|21 22|--- [DB5]
[DB6]     ---|23 24|--- [DB7]
Button1   ---|25 26|--- Button2
Button3   ---|27 28|--- Button4
Switch    ---|29 30|--- TempDir
Pressure1 ---|31 32|--- Pressure2
PwrSwitch1---|33 34|--- PwrSwitch2
          -------
[RS]      ---|35 36|--- [R/W]
PwrSwitch3---|37 38|--- PwrSwitch4
           ---|39 40|---
```

REFERENCES

Allen P. G., B. W. Louis, S. M. Anthonie, W. E. Johnston. 1984. Bioeconomics of Aquaculture, Elsevier, New York.

APHA, 1989. Standard Methods for the Examination of Water and Waste Water, 17th edition, American Public Health Association, American Water Works Association, Water Pollution Control Federation. Washington D.C.

Borland Inc. 1997, Borland C++Builder for Windows 95 and Windows NT. Borland International, Scotts Valley, CA.

BSO/Tasking. 1995. Software Development Under Control, Tasking Softwarte BV, Amersfoort, Netherlands.

Bray, D. R., P.A. Fowler, F. B. Fialho, R. A. Bucklin, S. P. Yeralan, T. Tran and R.K. Braun. 1998. An Automated System for Monitoring Milking System Partameters. In: Chastain, J. P., Ed. Proceedings of the Fourth International Dairy Housing Conference. ASAE Publication 01-98. American Society of Agricultural Engineers, St. Joseph, MI. pp17-24.

Bucklin, R. A., 1995. Automated Disposal of Effluent From Foodfish Production in Conjunction With Vegetable Production, Fifth Quarterly Report, Florida Energy Extension Service, University of Florida, Gainesville.

Carrow, R.S. 1998, Soft Logic, McGraw-Hill, New York.

CGIAR 1995. Consultative Group on International Agriculture Research. Press Release. http:/WWW.worldbank.org/html/cgiar/press/fish.html.

Ebeling, J. M. 1994. Aquaculture Water Reuse Systems: Engineering Design and Management, edited by Micheal B. Timmons and Thomas M. Losordo. Elsevier New York.

Ericsson, J. D. 1995. Case History of a Large-Scale Net-Pen Operation in the Gulf of Mexico: Challenges to Success. Aquacultural Engineering and Waste Management, Aquaculture in the Mid-Atlantic Conference, June 24-28,1995, Washington D.C.

FAO 1992. Aquaculture production. Food and Agriculture Organization of the United Nations, Rome, Italy. FAO Fisheries Circular, No. 815 Revision 4.

FAQ 1995. Microcontroller Primer. WWW.ba-karlsruhe.de/automation/FAQ/micro/index.html/.

Forrester, J.W. 1968. Principles of Systems, Wright-Allen Press, Inc. Cambridge, MA.

Froese, R. and D Pauly. 1997. FishBase 97: Concepts, Design and Data Sources, ICLARM, Manila, Philippines. FishBase 97 CD-ROM. ICLARM, Manila.

Greenberg, Jeff R. and J.R. Lakeland, 1997. A Methodology for Developing and Deploying Internet & Intranet Solutions, PrenticeHall, Upper Saddle River, NJ.

Hatch, U. and H. Kinnucan, 1993. Aquaculture Models and Economics, Westview Press, Inc. Boulder CO.

Kim, J., Y. K. Moon, and B. P. Zeigler, 1995. Designing Fuzzy Net Controllers Using Genetic Algorithms. IEEE Control Systems Vol 15 No. 3, pp 23-25.

Lee, P. G. 1994. A Review of Automated Control Systems For Aquaculture and Design Criteria for their Implementation. Elsevier Science Ltd. Aquacultural Engineering, Vol. 14, No 3, pp.205-227.

Losordo, T. M., J. E. Easley, P.E. Westerman. 1989. The Preliminary Results of a Feasibility study of Fish Production In Recirculating Aquaculture Systems. ASAE Winter Meeting, New Orleans.

Lawson, T. B. 1995, Fundamentals of Aquacultural Engineering, Chapman & Hall, New York.

MathSoft Inc. 1995. Mathcad PLUS 6.0. MathSoft Inc. Cambridge, MA.

Microchip. 1993. Embedded Control Handbook. Microchip Technology Inc, Chandler, AZ.

Motorola. 1996. Pressure Sensor Distributor Handbook, Motorola, Inc., Schaumburg, IL.

Netopia Inc. 1997. Virtual Office, Netopia, Inc., 2470 Mariner Square Loop,Alameda, CA.

Philips Semiconductors. 1995. Application Notes and Development Tools for 80C51 Microcontrollers, Sunnyvale, CA.

Philips Semiconductors. 1995. 80C51-Based 8-Bit Microcontrollers Data Handbook IC20, Sunnyvale, CA.

Pillay, T. P. 1990. Aquaculture Principles and Practices, Fishing News Books, Blackwel Scientific Publications Ltd. London.

Polyoarpov, M.M. and A. T. Vemuri, 1995. Learning Methodology for Failure Detection and Accommodation. IEEE Control Systems Vol 15 No. 3.

Random House. 1996. Random House Webster's Unabridged Electronic Dictionary, Random House Inc. New York.

Ricker, W.E., 1954. Stock and Recruitment. J. Fish. Res. Bd. Canada, 11:559-623.

Rusch, K. A. and R. F. Malone, 1993. A Micro-Computer Control and Monitoring Strategy Applied to Aquaculture. Techniques for Modern Aquaculture, American Society of Agricultural Engineers, St. Joseph, MI.

Savas, E. S. 1965, Computer Control of Industrial Processes , Mcgraw-Hill Book Company, New York.

Timmons, B. T. and T. Losordo. 1997. Advances in Aquaculture Engineering, Proceedings from the Aquacultural Engineering Society (AES) Technical Sessions at the Fouth International Symposium on Tilapia in Aquaculture, November 9-12, Orlando, FL, Northeast Regional Agricultural Engineering Service, NRAES-105, Ithaca, New York.

Timmons, B. T. and T. Losordo. 1994. Aquaculture Water Reuse Systems: Engineering Design and Management, Elsevier, New York.

von Bertalanffy, L. 1938. A quantitative theory of organic laws II. Hum Biol. 10:181-213

Wheaton, Fredrick W., 1993. Aquacultural Engineering, Krieger Publishing Company, Malabar, FL.

Whitson, J.L., P.E. Turk, C. Zuercher, and P.G. Lee. 1997. Advances in Aquaculture Engineering, Proceedings from the Aquacultural Engineering Society (AES) Technical Sessions at the Fouth International Symposium on Tilapia in Aquaculture, November 9-12, Orlando, FL, Northeast Regional Agricultural Engineering Service, NRAES-105, Ithaca, New York.

Yeralan, S.P. and A. Ahluwalia, 1995. Programming and Interfacing the 8051 Microcontroller, Addison-Wesley Publishing Company, New York.

BIOGRAPHICAL SKETCH

Philip A. Fowler was born on the 20th of April, 1952, in Bethesda, Maryland. The primary education was received in Alabama and Florida. The secondary education was gained at United States Merchant Marine Academy, Florida Keys community college, Tennessee Technological University, and University of Florida where he received a Bachelors of Science in agricultural engineering in 1978. He worked after that period for the next 12 years and then in 1990 went back to the University of Florida to pursue a Master of Engineering with the supervision of Howard Beck and then to pursue a Doctor of Philosophy, under the supervision of Dr. Ray Bucklin.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Ray A. Bucklin, Chairman
Professor of Agricultural and
Biological Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

C. Direlle Baird, Cochairman
Professor of Agricultural and
Biological Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Fedro S. Zazueta
Professor of Agricultural and
Biological Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Sencer P. Yeralan
Associate Professor Industrial and
Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Edward P. Lincoln
Professor of Agricultural and
Biological Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

December 1998

Winfred M. Phillips
Dean, College of Engineering

M. J. Ohanian
Dean, Graduate School